INTEGER APPROXIMATION OF REAL VALUED PREFERENCE CURVES

THESIS

Richard M. Antoine, Captain, USAF

AFIT/GLM/ENS/01J-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GLM/ENS/01J-01

INTEGER APPROXIMATION OF REAL VALUED PREFERENCE CURVES

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Logistics Management

Richard Antoine, B.S.

Captain, USAF

May 2001

AFIT/GLM/ENS/01J-01

INTEGER APPROXIMATION OF REAL VALUED PREFERENCE CURVES

Richard M. Antoine, B.S., M.S.
Captain, USAF

Approved:

Lt Col Alan W. Johnson, Advisor Assistant Professor of Logistics Management Department of Operational Sciences	Date
Major Stephen M. Swartz, Co-Advisor Assistant Professor of Logistics Management Department of Operational Sciences	Date

ACKNOWLEDGEMENTS

Thank you, wife, for patience and understanding. Thank you, daughter, whose mere presence reminded me what was really important. Thank you, mother, for teaching me perseverance. Thank you, father, for demonstrating a tireless work ethic. Thank you, God, for blessing me with all of them.

Richard M. Antoine

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

AFIT/GLM/ENS/01J-01

ABSTRACT

A primary challenge of the AFIT Mission Resource Value Assessment Tool is to approximate a given preference curve with integer valued mission ready resources. This thesis evaluated four candidate methods of accomplishing this approximation.

The thesis evaluated the implementation of the integer estimation approximation from a purely mathematical perspective. The models were measured against six quality and error measurement standards: convergence on an endpoint, convergence on any interior integer points, characterization of the overall error between the sequence of integer coordinates and the real valued linear function and characterization of the error in each individual dimension of the problem space. Finally, computer processing time was measured and a comparison of the lengths of the real valued linear function and the sequence of integer coordinates used to approximate the function were compared.

Based on these measures the Relative Slope Algorithm (RSA) was selected. RSA demonstrated the minimal error and consistently quick processing time. This algorithm will improve the Mission Resource Value Assessment Tool and further its impact on the Advanced Logistic project.

When you're up against a trouble meet it squarely face to face.
Lift your chin, set your shoulders, plant your feet and take a brace.
It may be vain to try to dodge, but do the best that you can do.
You may fail but, you may conquer see it through.

Black may be the clouds about you and your future may seem grim.
But, don't let your nerve desert you; keep yourself in fighting trim.
If the worst is bound to happen spite of all that you can do,
Running from it will not save you, see it through.

Even hope may be but futile when with troubles you're beset.
But, remember you are facing just what other men have met.
You may fail but fall still fighting, don't give up what e'er you do.
Eyes front, head high to the finish. See it through.

Edgar Guest

INTEGER APPROXIMATION OF REAL VALUED PREFERENCE CURVES

I. INTODUCTION

BACKGROUND

This research supports the Air Force Institute of Technology's contribution to the Advanced Logistics Project (ALP), which is sponsored by the Defense Advanced Research Projects Agency (DARPA). The Air Force Institute of Technology (AFIT) research contribution focuses on the development of the Mission Resource Value Assessment Tool (MRVAT) (Johnson, Swartz and Allen, 2000). The goal of this project is to improve deployment planning and execution for military combat forces. The research will result in an information technology system capable of integrating information from existing DoD logistics databases and producing a "best solution" to requested mission requirements. AFIT and the Air Force Research Laboratory (AFRL) have been asked to investigate some specific areas of the project (Buzo, 00:1). A critical goal of the ALP is to quickly develop mission solutions. Mission alternatives will be provided in a matter of hours instead of days (Buzo, 00:1). AFIT is developing the MRVAT software to demonstrate this aspect of the ALP technology. This research examines how the MRVAT can best assign Mission Ready Resources (MRRs) to an existing commander's stated mission preferences (represented by a real valued line function). An MRR is a resource configured for a particular purpose, such as an F-16 with a specified weapons and fuel load out (Swartz, 1999). Figure 1 illustrates a preference curve based on two competing tasks or missions. For example, in this scenario, the commander wishes to fly three daily Suppression of Enemy

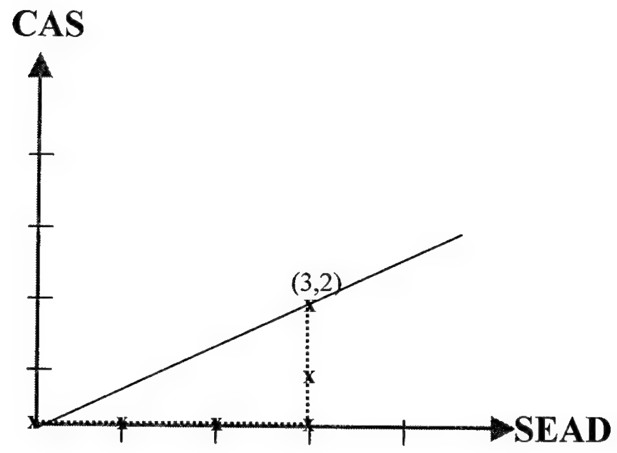

FIGURE 1: Preference curve with MRR assignments

(Johnson, Swartz and Allen, 2000)

Air Defense (SEAD) missions for every two Close Air Support (CAS) missions. We wish to allocate MRRs to missions one-at-a-time, to establish a priority order for deployment. Consider the coordinate (3,2) representing three daily SEAD missions and two CAS missions. We could sequentially allocate MRRs first to the three SEAD missions and then to the two CAS missions, as shown in Figure 1. However, nine other sequences can be constructed from the origin to the coordinate (3,2). Hence, which sequence is best? In other words, which sequence best preserves the commander's desired ratio of missions flown? A problem with several tasks and hundreds of assets would produce a much greater number of sequences. A method is required that can generate the sequence which most accurately approximates the task preference curve.

PROBLEM STATEMENT

What is the most efficient method of approximating a real valued linear function with a sequence of integer coordinates, where each coordinate is a unit distance from its adjacent coordinates? Note that for any two vectors $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, the distance between x and y is defined as $\|x - y\| = \sum_{i=1}^{n} |x_i - y_i|$. This research examines the method by which MRVAT and three alternate algorithms fit the integer number of resources available to the given real valued mission preference curve.

SCOPE

The research effort will review existing methods of plotting integer data with the intent of approximating a continuous and linear curve. Methods of determining the error between the sequence of integer coordinates and the continuous curve will also be examined. This research will be limited to the development of a mathematical research model. Actual MRRs and mission preferences will not be modeled. This will allow the model to draw from an unconstrained set of integers with which to map the continuous curve. The research results will present an effective, minimum error method of matching integers to a known continuous curve. The problem will be modeled in seven dimensions in order to demonstrate a robust multi-dimensional method that can be applied to the ALP MRVAT research effort. Note that other research (Wakefield, 2001) examines the optimization of MRR selection for a given set of mission preferences.

RESEARCH QUESTIONS

-What processes exist for approximating a real valued curve with a sequence of integer coordinates, each a unit distance from its adjacent coordinates?

-What methods will measure the error between a given real valued linear function and the sequence of integer coordinates?

-Can the problem be modeled in seven dimensions?

-What impacts will the research results have and how will they be used?

METHODOLOGY

Four methods of matching a sequence of integer coordinates to a real valued linear function will be evaluated. These algorithms will be modeled in Visual Basic and the resulting approximation will be compared to the real valued linear function. The error between the real valued linear function and the competing integer coordinate approximations will be characterized.

The model should converge on an end point, converge on any integer coordinates interior to the real line and the overall error between the sequence of integer coordinates and the real valued linear function should be characterized. The error in each individual dimension of the problem space should also be characterized. Finally, computer processing time shall be measured and a comparison of the lengths of the real valued linear function and the sequence of integer coordinates used to approximate the function will also be accomplished.

II. LITERATURE REVIEW

INTRODUCTION

Chapter II will address the focus of this research and discuss various methods which can potentially solve the problem. The chapter begins with an overview of the MRVAT. It then provides information on computational geometry, integer programming and line drawing algorithms. Once these subjects have been discussed, characteristics that may influence method selection are presented.

The MRVAT software created by the AFIT/ ALP research is essentially a technology demonstrator. The objective of the ALP research is to convert stated mission requirements into logistic needs. The MRVAT will provide theater commanders with the maximum combat capability possible within the logistical constraints (airlift) on transportation. The research focuses on an Air Expeditionary Force scenario with respect to solution generation (Johnson, Swartz and Allen, 2000).

This research attempts to discover an algorithm that can quickly generate a sequence of integer coordinates that accurately approximate a real valued linear function. The only link between this current effort and the AFIT/ ALP research is the similarity between the real valued linear function and the task preference curve described in Chapter 1.

METHODS REVIEWED

In order to solve the research problem, methods were sought to provide the approximation. Subjects that are reviewed and discussed include computational geometry, integer programming and line drawing algorithms.

Computational Geometry

Geometric design problems arise in many industries. “The unique designs of aircraft, cars and other modern machines require innovative ways of modeling the surfaces of these machines. In many cases a designer may draw a curve to approximately fit several points in some given plane” (Bu-Qing and Ding-Yaun, 1989:1). The formal definition of computational geometry is a "computer-based representation, analysis, synthesis (design) and computer-controlled manufacture of two and three dimensional shapes" (Bu-Qing and Ding-Yaun, 1989:1). In this situation, representation refers to the creation of a mathematical model, such as the equation of a line. Once the model is formed, information about the curve must be determined in order to evaluate points on the curve. Discovering unwanted loops or inflection points are activities that fall under synthesis and analysis.

The most applicable computational geometry method is range search. Range-search problems are of particular interest with respect to our thesis problem. Range-search problems require that a collection of points be represented in a space. In this case the query is a space in which a set of points reside. The query space can be described as a standard geometric shape (i.e. ball or box). The range search is essentially the retrieval or counting of all points in the query space (Shamos and others, 1990:40).

Integer Programming

Integer programming (IP) has a wide range of applicability. This method is used when modeling the use of resources that logically must be represented by integer numbers (i.e. airplanes, cars or houses). For certain items, it does not make sense to have a fractional amount of that resource.

IP's have great risk associated with regard to problem solving. Mathematically, IP models require much more computation time than a similarly sized linear program (LP). There is a great possibility that an IP model may not be solvable in a reasonable time period. Consider that in comparison to an LP that can be rapidly solved with thousands of constraints and variables (Williams, 1985).

Sometimes it is difficult to determine when an IP is applicable. In this section we will discuss some of the types of problems IP's can solve.

The most obvious case was previously mentioned. When a problem requires whole numbers of products or uses integral units of a resource, it is a problem with discrete inputs and outputs. An IP formulation would be appropriate for these situations.

Many problems have a large number of feasible solutions arising from different orders of performing operations or the allocation of items to certain positions. These types of problems are called combinatorial problems. This category can be further divided into sequencing problems and allocation problems.

Sequencing would take the form of a scheduling operation, Job-Shop scheduling or the optimal ordering of operations on different machines in a Job-Shop (Williams, 1985). Another example of sequencing is the traveling salesman problem. This problem seeks a solution that describes the optimal order in which to visit a number of cities and return home within the minimum distance.

The market share problem is a good example of an allocation problem. The allocation of customers to divisions in a company depending on service is the objective of this problem. Another example of an allocation problem is the assembly line balancing problem. The goal of this problem is to assign workers to tasks in order to achieve a certain rate of production.

Our problem requires a method that will minimize the distance between the coordinates used for approximation and the real valued linear function. An integer program with a minimizing objective function may be appropriate.

Line Drawing Algorithms

The subject of line drawing comes from a study of computer graphics models. Line, circle and surface drawing are methods by which lines and other geometric objects are properly represented in graphics. "Incremental computing techniques are a form of iterative computation, in which each iterative step is simplified by maintaining a small amount of state, or memory, about the progress of the computation" (Newman & Sproull, 1979:19). Incremental methods are very useful due to their simplicity and accuracy. These methods allow the user to determine which pixels on the computer screen to illuminate, and ultimately provide the most exact graphical representation of the line.

There are three important characteristics a line drawing algorithm must have in computer graphics. The line must appear straight, terminate accurately and have constant density (Newman & Sproull, 1979:21). A line can have a well defined end point and start point, while its entire set of interior points do not pass through any integer coordinates. If an algorithm approximates the interior points accurately the line will appear straight. Lines must be plotted accurately in order to prevent gaps between the end point of one line and the start point of the following line. Density is important so that the line has a consistent resolution from beginning to end (no light spots). These characteristics are directly applicable to the MRVAT problem. We do not want our sequence of integer coordinates to wander through the problem space as the solution is approximated. We wish the approximation to appear straight, and to closely match the real line being approximated. Our algorithm must also terminate accurately if it is to converge on the solution (end point) that we have selected. Finally, density is also important. The algorithm must converge on the solution in one unit increments.

Two methods of line drawing that are directly applicable to this thesis problem were found. These are the Digital Differential Analyzer (DDA) algorithm and Bresenham's Line Drawing Algorithm (BLDA).

Before discussing DDA we shall review a few basic concepts. The slope intercept equation describing a straight line is:

$$y = m \cdot x + b \tag{1.1}$$

where m is the slope and b is the y intercept. For any two coordinate pairs (x_1, y_1) and (x_2, y_2), we have slope m and intercept b defined in equations 1.2 and 1.3:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{1.2}$$

$$b = y_1 - m \cdot x_1 \tag{1.3}$$

These simple equations provide the basis for algorithms that display straight lines in two dimensional space. For a given x-interval (Δx) on a line we can compute the y-interval (Δy) from equation 1.2.

$$\Delta y = m \cdot \Delta x \tag{1.4}$$

Equation 1.5 shows us how to obtain the x-interval.

$$\Delta x = \frac{\Delta y}{m} \tag{1.5}$$

The DDA works on the principle that we simultaneously increment x and y in small steps proportional to Δy and Δx (Newman & Sproull, 1979:22-23). The first derivatives are constant and proportional to Δy and Δx when considering a straight line (Newman & Sproull, 1979:23). A sample of the line is taken at unit intervals in one direction and a calculation of the corresponding integer values nearest the line path if determined for the other direction (Hearn & Baker, 1997:86). The simple DDA requires that a line-length estimate be set equal to the larger of the magnitudes of Δx and Δy. This allows the increment value in the x or y direction to be of unit magnitude. This allows unit steps to be made in the direction of steepest ascent. We assume that m lies between 0 and 1. If m $(\Delta y/ \Delta x)$ is less than 1 then a 1 unit increment is made in the x direction and a rounding calculation is made in the y direction based on the slope. This assumes that the magnitude of x is greater than y. Successive y values are now determined by:

$$y_{k+1} = y_k + m \tag{1.6}$$

The subscript k takes on integer values starting with one for the first point and increases by one until the end point of the line is reached (Hearn & Baker, 1997:86-87). If the line has a positive slope ($\Delta y > \Delta x$) then the roles of x and y are reversed. In this case the y direction is incremented by 1 integer unit and the rounding calculation is made in the x direction based on the slope.

$$x_{k+1} = x_k + \frac{1}{m} \tag{1.7}$$

Equations 1.6 and 1.7 both operate on the assumption that the line will be approximated from the left end point (origin) to the right end point (solution point). In the case of approximating the line from right to left, the following modifications to these equations would be made:

$$\Delta x = -1 \ldots, y_{k+1} = y_k - m, \tag{1.8}$$

$$\Delta y = -1 \ldots, x_{k+1} = x_k - \frac{1}{m}. \tag{1.9}$$

When the slope of the line is negative then take the absolute value of the slope ($|m|$). Equations 1.6 through 1.9 would be used as previously described (Hearn & Baker, 1997:86-87).

Bresenham's line algorithm converts lines into graphical representations using incremental integer calculations that can be adapted to display circles and other types of curves. This algorithm also assumes that slope must be less than 1 ($m < 1$). BLDA also requires that the line be approximated from the left endpoint to the right endpoint. Incremental steps are made for each x position and the point whose y value is closest to the curve is plotted. Once point (x_k, y_k) is determined, then the next appropriate point to plot must be found. There are two choices at this point, ($x_k + 1, y_k$) or ($x_k + 1, y_k + 1$). The vertical distances of the potential points at x_{k+1} are called d_1 and d_2. These points describe the vertical point separations from the given curve. The y coordinate on the curve will be calculated as follows:

$$y = m \cdot (x_k + 1) + b \tag{1.10}$$

$$d_1 = y - y_k = m \cdot (x_k + 1) + b - y_k \tag{1.11}$$

$$d_2 = (y_k + 1) - y = y_k + 1 - m \cdot (x_k + 1) - b\,. \tag{1.12}$$

The following is the resulting difference equation:

$$d_1 - d_2 = 2 \cdot m \cdot (x_k + 1) - 2 \cdot y_k + 2 \cdot b - 1\,. \tag{1.13}$$

A decision parameter p_k at the *k*th step in the algorithm can be found by manipulating equation 1.13 until it only involves integer calculations. This is achieved by substituting $m = {}^{\Delta y}\!/_{\Delta x}$ in place where Δy and Δx are the vertical and horizontal endpoint positions. The resulting equation is as follows:

$$p_k = \Delta x \cdot (d_1 - d_2) = 2 \cdot \Delta y - x_k - 2 \cdot \Delta x \cdot y_k + c\,. \tag{1.14}$$

If it turns out that the point at y_k is closer to the curve than the point at y_{k+1} $(d_1 < d_2)$ then this makes the sign of p_k negative. In this situation we would like to plot the lower point or else plot the upper point. Coordinate changes along the line occur in incremental integer steps in either the *x* or *y* directions. This means that the values of the successive decision parameters can use incremental calculations. At step $k+1$ we can use equation 1.14 with the following adjustments:

$$p_{k+1} = 2 \cdot \Delta y \cdot x_{k+1} - 2 \cdot \Delta x \cdot y_{k+1} + c\,. \tag{1.15}$$

Now subtract equation 1.14 from 1.15 to get the following:

$$p_{k+1} - p_k = 2 \cdot \Delta y \cdot (x_{k+1} - x_k) - 2 \cdot \Delta x \cdot (y_{k+1} - y_k) \tag{1.16}$$

and now by substitution $x_{k+1} = x_k + 1$

$$p_{k+1} = p_k + 2 \cdot \Delta y - 2 \cdot \Delta x \cdot (y_{k+1} - y_k) \tag{1.17}$$

where $y_{k+1} - y_k$ is either 0 or 1 (Hearn and Baker, 1997:88-90). This recursive calculation for the decision parameters is accomplished at each integer x position. The calculation is made from the left endpoint to the right endpoint. To begin the algorithm, the parameter p_0 is calculated using equation 1.18. The start point at (x_0, y_0) is used in the slope equation ($m = \Delta y / \Delta x$). The following starting equation results:

$$p_0 = 2 \cdot \Delta y - \Delta x. \tag{1.18}$$

The BLDA uses decision parameters to determine the best axis direction to increment. Equation 1.18 is used to calculate the initial decision parameter. The decision parameter p_k must be evaluated. If p_k is less than zero ($p_k < 0$) then equation (1.19) is used. The decision also requires the next coordinate plotted be, $(x_k + 1, y_k)$.

$$p_{k+1} = p_k + 2 \cdot \Delta y \tag{1.19}$$

If p_k is greater than or equal 0 ($p_k \geq 0$) then equation (1.20) is used. This decision requires that the next coordinate plotted be $(x_k + 1, y_k + 1)$.

$$p_{k+1} = p_k + 2 \cdot \Delta y - 2 \cdot \Delta x \tag{1.20}$$

The decision parameter p_0 is only used for the initial decision. Once p_0 is compared to zero and a coordinate set is chosen for the next increment, equations 1.19 and 1.20 are used for the remainder of the increment decisions until the solution point is reached (Hearn and Baker, 1997:88-90).

MRVAT Integer Estimation Algorithm

The approach that MRVAT currently takes to incrementing integer values uses a relative slope approach. The difference between the origin (0,0) and end point (3,2) of the line (real valued linear function) are inspected for the largest delta value. The largest delta value becomes the denominator for the slope calculations. The axis values (preferences) are incremented by integer units according to the order of largest slope value to lowest slope value. Table 1 contains the start point (0,0) and the end point (3,2), and also provides the corresponding delta values.

TABLE 1: MIEA DELTA VALUE COMPARISON

x	y
0	0
3	2
$\Delta x = x_2 - x_1 = 3$	$\Delta y = y_2 - y_1 = 2$

The current MRVAT approach selects Δx as the largest delta value. The Δx and Δy values are each divided by Δx. This produces the following relative slope values: $\Delta x/\Delta x = 3/3$ and $\Delta y/\Delta x = 2/3$. The relative slopes dictate the order in which the approximation increments. The axis value with the higher slope is incremented until the corresponding end point is reached. In this example the x-axis value has the higher slope value (m=1). The approximation increments from 0 to 3 in the x direction. The y axis has the next highest slope (m=2/3). The approximation increments in the y direction accordingly. The algorithm increments from 0 to 2 in the y direction. This series of increments brings us to the solution point (3,2). The coordinate sequence is shown in Table 2.

TABLE 2: MIEA COORDINATE SEQUENCE

Program Output	*1*	*2*	*3*	*4*	*5*
x	1	2	3	3	3
y	0	0	0	1	2

The MIEA converges upon the solution by incrementing along each axis direction until the end coordinate in that direction is reached (Thomas, 2001). This series of coordinates takes the longest route possible along the exterior faces of the query space to the solution. The coordinate sequence is further described in Figure 2.

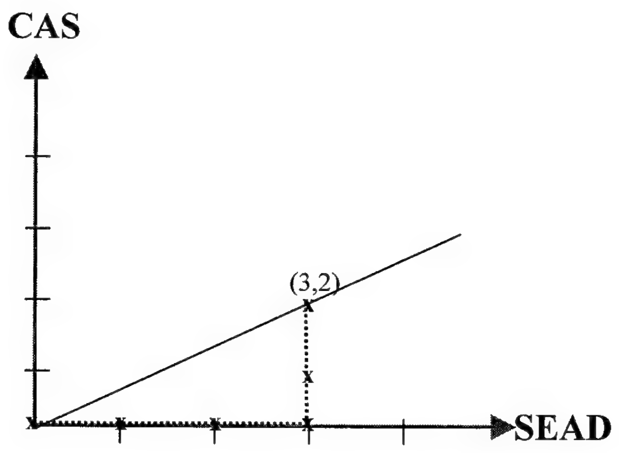

FIGURE 2: MIEA APPROXIMATION

Relative Slope Algorithm

The Relative Slope Algorithm (RSA) is a line approximation method developed by AFIT researchers in support of MRVAT (Swartz, 2001). This approach is also based on a form of slope comparison. The difference between the start point and end point in each dimension is calculated. The largest delta value is used as the denominator for slope calculations in all directions (see Table 3).

TABLE 3: RSA DELTA VALUE COMPARISON

x	y
0	0
3	2
$\Delta x = x_2 - x_1 = 3$	$\Delta y = y_2 - y_1 = 2$

The example in Table 3 leads us to use the Δx value as the denominator for the slopes calculated for each axis direction ($m1$ = 3/3, $m2$ =2/3). Slopes $m1$ and $m2$ tell us how to increment in x and y in order to approximate the line with integer values. The starting x coordinate will be incremented by units of one until it reaches the end point. The starting x coordinate will be incremented by the rounded (RND) value of its slope until it reaches its respective endpoint. See the following equation 1.21 (Swartz, 2001):

$$P_n = \left(x_o + RND\left(n \cdot m_x\right),\ y_o + RND(n \cdot m_y),\ z_o + RND\left(n \cdot m_z\right)\right) \qquad (1.21)$$

Equation 1.21 describes how any point $P_n = (x_n, y_n, z_n)$ is calculated. In this equation the starting point is identified by the coordinate (x_o, y_o, z_o). The relative slopes are identified by m_x, m_y and m_z. The value of n is equal to the number of points or increments away from the starting point. The n value is incremented by one with respect to each increment from the origin in each axis direction until the incremental values equal the end point coordinate values. The RSA approach requires that only one increment be made at a time in any direction. The unit steps will follow the order of steepest to shallowest slope ($m1$, $m2$) (Swartz, 2001). Table 4 demonstrates the first pass of the algorithm in which the relative slopes are added until the solution point (3,2) is reached.

TABLE 4: RSA SLOPE INCREMENT

Program Output	*1*	*2*	*3*
x	1	2	3
y	0.667	1.333	2

Once the RSA has made all of its increments and converged on a solution, it rounds the incremented values and back fills all the gaps that may exist between the points it has identified (Swartz, 2001). This is illustrated in Tables 5 and 6.

TABLE 5: RSA ROUNDED COORDINATE VALUES

RD Table	*1*	*2*	*3*
x	1	2	3
y	1	1	2

The algorithm must now fill in the increment steps between the rounded coordinate values. The increment will follow the order of steepest to shallowest slope as mentioned above. In this example the *x* direction will be incremented once and then the *y* direction. This order will be used between each coordinate starting with the origin. The final filled in coordinate sequence is provided in Table 6.

TABLE 6: RSA COORDINATE SEQUENCE

Fill In Table	*1*	*2*	*3*	*4*	*5*	*6*
x	0	1	1	2	3	3
y	0	0	1	1	1	2

Table 6 demonstrates the use of the relative slope between the rounded coordinates. The algorithm increments from the (0,0) to (1,1), from (1,1) to (2,1), and from (2,1) to (3,2). Between each coordinate it increments in the *x* direction and then the *y* direction according to the relative slopes previously identified. The example is graphically depicted in Figure 3.

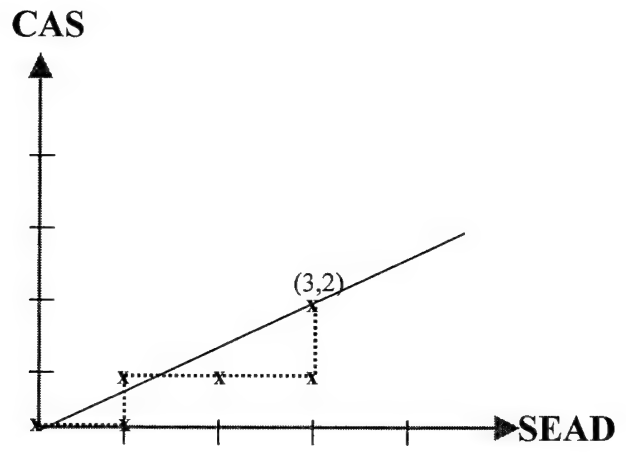

FIGURE 3: RSA Approximation

The RSA method fills our requirements by incrementing in each axis direction by one unit at a time. The algorithm performs division and rounding operations on its first two passes (Table 4, Table 5). These operations can potentially slow the computer processing. Further analysis of the algorithm will be discussed in Chapter IV.

Continuous Segment Algorithm

The Continuous Segment Algorithm (CSA) follows the same logic as the RSA. The difference between the two algorithms lies in the manner in which the slopes are determined. The CSA uses the product of the largest delta value and the number of dimensions that define the problem space. This number can be referred to as an alpha value. All deltas in each dimension are divided by the alpha value (i.e., $\alpha = (\Delta y)$.(7 dimensions)) (Swartz, 2001). This provides the modified slope that is ultimately used to increment the values in their respective dimensions. Table 7 demonstrates the procedure.

TABLE 7: CSA ALPHA VALUE CALCULATION

x	y
0	0
3	2
$\Delta x = x_2 - x_1 = 3$	$\Delta y = y_2 - y_1 = 2$
$\alpha = \Delta x \cdot 2\,Dimensions$	$\alpha = (3)(2) = 6$

The example in Table 7 demonstrates the calculation of the alpha value using the largest delta value and the number of dimensions. The alpha value is then used to calculate the relative slopes that determine order in which each axis direction will be incremented. The slopes (m1 = 3/6, m2 = 2/6) are used to increment from the start point (0,0) to the end point (3,2) as demonstrated in Table 8.

TABLE 8: CSA SLOPE INCREMENT

Program Output	*1*	*2*	*3*	*4*	*5*	*6*
x	0.5	1	1.5	2	2.5	3
y	0.333	0.667	1	1.333	1.667	2

Similarly to the RSA, once the CSA has made all of its increments and converged on the solution, it rounds the incremented values and eliminates any duplicate points that are created during the rounding procedure (Swartz, 2001). Table 9 contains the results of the rounding procedure.

TABLE 9: CSA ROUNDED COORDINATE VALUES

RD Table	*1*	*2*	*3*	*4*	*5*	*6*
x	1	1	2	2	3	3
y	0	1	1	1	2	2

The algorithm must now eliminate any duplicate coordinates and simultaneously fill in increment steps where required. Once a coordinate has been recorded at this step it is not repeated which results in the elimination of duplicates. If a gap is identified then it is filled by incrementing in the appropriate axis direction. In our example the *x* direction is incremented first and the *y* direction is incremented second according to the order of the relative slopes. The final fill in and duplicate elimination coordinate sequence is provided in Table 10.

TABLE 10: CSA COORDINATE SEQUENCE

Fill In Table	*1*	*2*	*3*	*4*	*5*	*6*
x	0	1	1	2	3	3
y	0	0	1	1	1	2

We can also compare Table 9 with Table 10 and observe where the duplicate points were eliminated and additional points were added. Unit increments were made between rounded coordinate (2,1) and rounded coordinate (3,2). The duplicate rounded coordinates of (2,1) and (3,2) are eliminated during the final step. The example is graphically depicted in Figure 4.

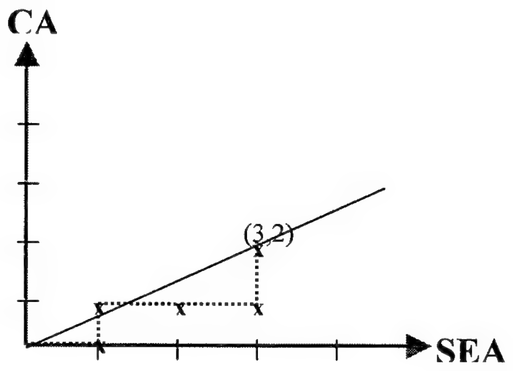

FIGURE 4: CSA Approximation

It should be noted that in this case, the point between (2,1) and (3,2) was "filled" using the same method as RSA. If the α value were made larger (resulting in smaller increments and more duplicates), the RSA "fill method" would not be required. This is potentially a subject for future exploration.

METHOD SELECTION

Computational geometry provides location and range-search approaches to finding points. These approaches require that the solution space be partitioned and the integer points found or identified. This approach does not meet the requirement for a one unit increment towards the solution. Our problem does not require that all the points be identified in a space. This would be very inefficient. The points that keep the sequence on integer coordinates within a minimum distance of the real valued linear function will provide the best answer.

Integer programming provides a method of minimizing an objective function according to a set of constraints while constraining the problem to integer solutions. Our problem requires that the distance be minimized between the selected integer points that are used in the sequence of integer coordinates and the real valued linear function itself. This approach loses its applicability when the constraints must be selected. The limitations on our problem do not lend themselves to an integer program problem. The increments must happen in unit steps as stated above but, the number of units is unconstrained.

Bresenham's Line Drawing Algorithm (BLDA) is designed for a two dimensional problem and it allows incremental moves in the x and y directions. BLDA requires the use of decision parameters for determining how far to move in each dimension. This would create a messy transition from a two dimensional problem to a n-dimensional problem. BLDA's incremental calculations are integer based and do not rely on floating point calculations or rounding functions for any of its decision parameters or unit increments. This algorithm specifically benefits the graphic designer. The integer based calculations increases the speed at which the program can draw the line if not the accuracy.

The Digital Differential Analyzer (DDA) is also designed to operate in a two dimensional space. Once the slope comparison portion of the algorithm indicates which slope combination is increasing at a greater rate, it then begins incrementing. The algorithm increments by unit steps in the direction of steepest ascent that is less than or equal to one. It increments in the other direction by adding the slope value and rounding that value to a whole number. This second increment involves a floating point calculation (division to find slope). This will cause the DDA program to run slower than the BLDA program. However, the BLDA algorithm does not provide a more accurate solution than the DDA. "Sproull has shown that the Bresenham algorithm can be derived from the differential analyzer, thus establishing that both generate identical output moves" (Earmshaw, 1985:135).

The current MRVAT Integer approximation method also uses the slope to determine which axis direction to increment. This approach satisfies our need to perform only one unit step at a time as opposed to the DDA and BLDA algorithms which increment in both directions simultaneously. The current MRVAT approach is also capable of providing an n-dimensional solution. However, it requires that each direction chosen must be incremented until the endpoint is reached in that direction. This may lead to error in the approximation.

The relative slope method requires that a unit step increment be made in the axis direction of steepest accent. Again, the direction is based on a comparison of the slopes. The other axis directions are incremented according to the slope with respect to that axis direction. This requires that a floating point and rounding calculation be made in the remaining axis directions. In addition to slowing the program down, there will be an opportunity for rounding error. RSA also requires that calculations be made in order to fill in any gaps between the increments that it has made on its first pass.

The CSA approach is subject to the same pros and cons as the RSA. An additional "con" of the CSA is its method of line segmentation. The CSA algorithm divides its real valued linear function into much smaller segments than the RSA. This is due to the alpha value used as the divisor (i.e. $\Delta y/\alpha$) tends to be much larger than the largest relative slope value used by RSA. This increases the number of addition and rounding operations that must be performed. If there is a significant amount of extra work, there may be a significant amount of extra computation time.

Ultimately, the method that is selected must allow us to solve a multi-dimensional problem while taking only one unit step at a time towards the solution. The approach must also meet our five quality/error measurements stated in the methodology section of chapter one. Another potential approach based on the DDA algorithm will be discussed in Chapter III.

CONCLUSION

Chapter II provided a general description of how the MRVAT program requires integer approximations to a real line and provided a basis for the thesis problem. Information was presented on computational geometry, integer programming and line drawing. The pros and cons of these methods were also discussed with respect to our thesis problem. The next chapter will discuss the development of the Slope Comparison Algorithm, which was developed by the thesis author. The error measurement methods will also be described. Finally, the experimental design will be detailed.

III. METHODOLOGY

INTRODUCTION

Chapter III will discuss the development of the Slope Comparison Algorithm, which draws some of its evaluation rules from the Digital Differential Analyzer. The error calculation method will also be presented. The chapter ends with a discussion of the experimental design for this research.

DDA Transformation

The first step in this process is converting the given two dimensional version of the Digital Differential Analyzer to a multi-dimensional algorithm. The DDA algorithm uses the largest value of *x* or *y* and uses that number as the denominator of the slope calculation. For example, a large *y* value indicates a faster increase in the *y* direction versus the *x* direction. The slope must be between 0 and 1 ($0 \le m \le 1$) (Jaccobs, 2001). The denominator variable of the slope is incremented by one integer step and the numerator variable of the slope is incremented by a rounded value of the slope itself. In the following example the slope is $\Delta y / \Delta x$ (Hearns and Baker, 1997:88):

$$y_{increment} = y_0 + round(m)$$
$$x_{increment} = x_0 + 1$$

In order to approximate a line in seven dimensional space, we must consider slopes based on seven axes. The axes for this problem are labeled *x*, *y*, *z*, *t*, *u*, *v*, and *w*. The slope calculation can only be made with points from two of the dimensions at a time. Our problem will compare each possible slope value versus dividing by the largest axis coordinate value. The number of dimensions tells us that a "seven choose two" permutation must be used in order to determine the number of potential slope combinations. Any ordered sequence of k objects taken from a set of n distinct objects is called a permutation of size k of the objects. In other words, permutations are used when order matters. The order of the coordinates used in the problem space matters because there is a difference between x/y and y/x (Devore, 2000:70).

$$P_{k,n} = \frac{n!}{k!(n-k)!} \tag{1.22}$$

Equation 1.22 is used to make the permutation calculation. Forty-two slope combinations are necessary in order to properly consider this seven dimensional problem. Once the DDA algorithm is used to decide which axes to increment, both the axes in the two – dimensional problem are incremented simultaneously as indicated in the above DDA example. Our problem requires that only one integer unit step be taken in the axis of choice at a time. The program compares the forty-two slope calculations based on the DDA criteria requiring that the slope fall in between 0 and 1 ($0 \leq m \leq 1$) and the slope value must also be the closest to one. A unit increment is made in the axis direction associated with the variable in the denominator. This axis is the only direction that is incremented per iteration. The appropriate axis values are updated with the new coordinates and all the slopes are re-calculated based on the last step (the step is used as the new starting point in the slope calculations). All forty-two new slopes are compared and the next increment is made based on the criteria previously described. The program goes through this routine until the values (coordinates) associated with each axis are equal to the endpoint of the line. Our transformed DDA will be referred to as the Slope Comparison Algorithm (SCA). Table 11 provides an example starting and ending point.

TABLE 11: SCA POINT VALUES

x	*y*
0	0
3	2

The Slope Comparison Algorithm's user would enter the defining points of the line that will be approximated. Table 12 demonstrates a sequence of integer coordinates used to approximate the line defined by the input coordinates in Table 11.

TABLE 12: SCA COORDINATE SEQUENCE

Axis	*1*	*2*	*3*	*4*	*5*	*6*
x	0	1	1	2	2	3
y	0	0	1	1	2	2

Based on slope comparisons, each axis direction is incremented by one unit until the ending point is reached. All possible slopes are calculated based on the difference between the end point and the current starting point. The slope is judged on the criteria described earlier and the increment is made based on that decision. The starting point is equal to the coordinates identified at each previous iteration. The initial starting point is obviously, the point (0,0) originally entered. The next starting point is (1,0). This process progresses until the solution is reached. Figure 5 provides a graphical depiction of the sequence described in Table 12.

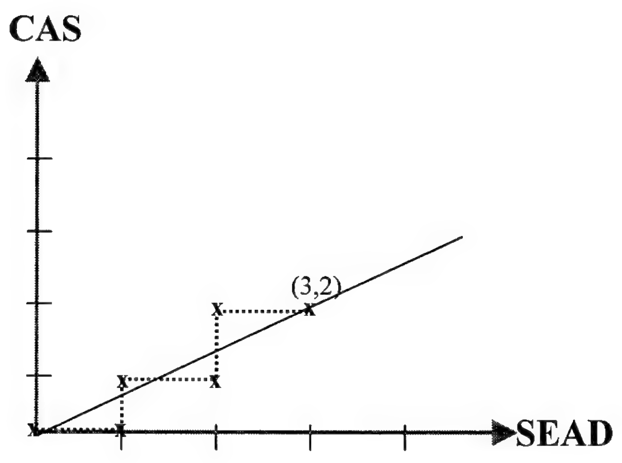

FIGURE 5: SCA Approximation

Once the SCA algorithm has approximated the curve and converged on the solution we must ascertain the quality of the fit. This will be accomplished by using a parametric distance equation. The process will be discussed in the next section.

Error Calculation

We shall now recall the six quality measurements that are required to validate the most efficient line approximation algorithm among the given algorithms. The model should converge on an end point, converge on any interior integer points and the overall error between the integer approximation and the real line (preference curve) should be characterized. The error in each individual dimension of the problem space should also be characterized. The computer processing time measurement and a comparison of the lengths of the integer line with the real line will also be made.

Convergence on the model solution point and interior points will be verified by running a series of tests using the experimental problem and other test information discussed in the experimental design section. The computer processing time can be measured by modifying the source code to report the elapsed time between the start and end of a given program. The remaining quality criteria can be evaluated using a parametric distance equation and some other variations in the source code.

A parametric distance equation is required for the distance calculation because of the multiple dimensions in our problem. The distance calculation is made from the point on the real line that is perpendicular to the off curve integer point used to approximate it. "The distance from a point to a line is defined as the minimum of all distances from the point to points on the line. This minimum will occur when the line from R to the point on the line is perpendicular to the line" (Foley and others, 1990:1100). The problem can be further described by Figure 6 and equation 1.23.

$$(R - P(t)) \cdot v = 0 \tag{1.23}$$

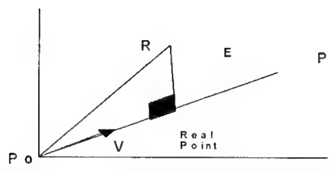

FIGURE 6: Parametric Equation Example

(Baker, 2001)

In equation 1.23 the function *P*(*t*) is subtracted from the off curve integer point R and multiplied by the vector *v*. The difference between *R* and *P*(*t*) will provide us with the distance *E*. The equation must be set equal to zero in order for the solution E to be perpendicular to the real line P_o to P_1. The variable *t* represents the distance from the origin (start point) of the line while *v* defines the direction of the line. *R* represents the off curve integer coordinate which is produced by the integer approximation.

$$t = \frac{(R - Po) \cdot v}{v \cdot v} \tag{1.24}$$

Equation 1.24 can be derived from equation 1.23 by substituting P_o +*tv* in place of *P*(*t*) (Foley and others, 1990:1100). After we compute the value of *t* it must be substituted into equation 1.25.

$$P_o + t \cdot v = 0 \tag{1.25}$$

In our problem *v* is defined as P_1 - P_o (Baker,2001). The value *t* is multiplied times *v* forming a new vector *tv* which is added to the vector P_o. Equation 1.25 ultimately provides us with the point on the real valued linear function as seen in Figure 3 (real line point). The minimum distance *d* between the off curve integer point *R* and the real line point can be calculated using equation 1.26.

$$d = \sqrt{(R_1 - RP_1)^2 + (R_2 - RP_2)^2 + (R_3 - RP_3)^2} \tag{1.26}$$

R is a vector which contains the coordinates of the off-curve integer point. *RP* is short hand for real line point. *RP* represents each of the coordinates that represent the point on the real line. The parametric distance equation is used to find the point on the real valued linear function (real line) which is the minimum (parallel) distance from the off-curve integer point. The distance equation is then used to calculate the magnitude of the distance between the points identified. This procedure is repeated for every integer point the algorithm creates as it determines the sequence of integer coordinates used to approximate the given real valued linear function.

The distance calculations made at each point provides us with an average distance between the sequence of integer coordinates and the real valued linear function. This information will characterize the overall error between the two lines. The error in each dimension is described by considering the average of the distances at each point in only one dimension at a time. Table 13 provides us with sample output of the error calculation program. PoCurve*k* represents the actual point (*x*,*y*) on the real valued linear function which is perpendicular to the off-curve integer coordinate produced by the integer approximation. The variable *d* represents the distance between those two points. The distance in each single dimension is represented by *x* distance, *y* distance, etc. For example, this is just the distance for between integer coordinate (*x*) and real coordinate (*x*). The real valued linear function length is represented by the term Real Line Distance and Integer Line Distance represents the length of the sequence of integer coordinates.

TABLE 13: SCA ERROR CHARCTERIZATION OUTPUT

Axis	*1*	*2*	*3*	*4*	*5*
Po Curve1	0.693	1.154	1.846	2.307	3
Po Curve2	0.462	0.769	1.231	1.538	2
d	0.5547	0.2774	0.2774	0.5547	0
d sum	1.664	davg	0.3328		
	1	2	3	4	5
x distance	0.3077	0.1538	0.1538	0.3077	0
y distance	0.4615	0.2308	0.2308	0.4615	0
Real Line Distance	0.8321	1.3868	2.2188	2.7735	3.6055
Integer Line Dist.	*1*	2	3	4	5

The final error that must be calculated is the difference between the Real Line Distance and the Integer Line Distance. The distance equation can be used to measure the distance from the origin to the end of the real valued linear function as shown in equation 1.27.

$$d = \sqrt{(R_n - P_o)^2} \qquad (1.27)$$

The sequence of integer coordinates (integer line) cannot be calculated as a straight line. In order to obtain the approximated numbers the algorithm must move along the adjacent and opposite sides with respect to the real valued linear function (real line). For each iteration of the problem there is a number of such moves which are made to obtain the off-curve integer coordinate. In the SCA the number of moves are restricted to one unit movement per iteration. This means the integer distance increases by one unit per iteration. In the last line of Table 13 we observe the increasing value of the integer until it reaches five. For our problem, the integer line distance is simply an addition of all the unit moves made by the algorithm as it converges on the solution point (end point). Figure 7 illustrates the error measurements along the curve.

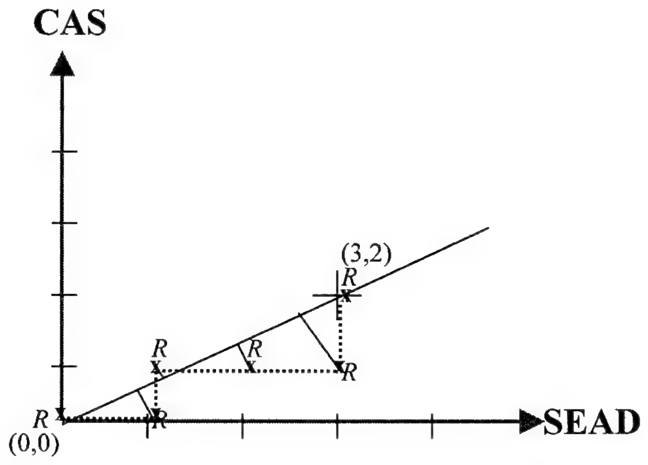

FIGURE 7: Error Measurements on Preference Curve Example

The real valued linear function (real line) is defined by the coordinates (0,0) to (3,2). The sequence of integer coordinates are represented by the dashed line. The integer coordinates are labeled as *R*. The line extending from each coordinate *R* to the real line is the perpendicular distance between the off-curve integer coordinate (*R*) and the coordinate on the real line. Note that the origin and endpoint fall on the solid real line and the dashed integer line. The distance is measured at each of the five integer coordinates used to approximate the real line. The length of the real line is measured by performing a distance calculation using the coordinates (0,0) and (3,2). The length of the integer line is equal to the five segments of the dashed line that begins at point (0,0) and ends at (3,2).

The source code for the Slope Comparison Algorithm can found in Appendix F. Now we will examine the set up necessary for comparing the four candidate algorithms and determining the error and quality of each.

Experimental Design

We will now consider the evaluation of four approaches to the integer estimation of a preference curve. Each of the methods has been coded using Microsoft Visual Basic 6.0. The programs were run on a Dell Inspiron 7500 laptop computer. The Excel program being run was the only active application during the experiment. These measures ensured that the computer resources were devoted to running the program. The programs for each method were created to interact with an Excel spreadsheet. The programs pull data from and send data to the spreadsheet as they approximate the given line. The starting and ending coordinates must be provided for each programmed method. The MIEA, RSA and CSA methods all require the number of dimensions to be entered. The four approximation methods assume that the line is being approximated from the left hand coordinate to the right hand coordinate. It was also assumed that the ending points will be higher in magnitude to their corresponding starting points. Each of the four programs were limited to a maximum of seven dimensions. The solution data was recorded in the Excel sheet by row. Excel limited our solutions to 32,767 rows due to the integer defined counters in the programs.

The MIEA, SCA, RSA and CSA methods were evaluated on the basis of the six quality and error measurements described in Chapter I. The model should converge on an end point, converge on any interior integer points and the overall error between the sequence of integer coordinates and the real valued linear function (preference curve) was characterized. The error in each individual dimension of the problem space should also be characterized. The computer processing time and a comparison of the lengths of the integer line with the real line were also be made.

Convergence on an endpoint and interior points are simply a matter of observation and reporting. As the algorithms were evaluated with various data sets, it was determined whether these first two quality criteria are met. The remaining four criteria must be evaluated based on a formal test plan.

The experiment evaluated the algorithms using 3 dimensional, 5 dimensional, and 7 dimensional problem spaces. In each problem space, three sets of start and end points were identified. The first set had an end point with a high difference between individual variable slopes. The second set had a medium distance in slope and the third set had no difference in slope. Each algorithm was tested using these same sets (start and end points) entered into the software and run to a solution. The test points are presented in Table 14.

TABLE 14: TEST SOLUTION END POINTS

Dimensions	High Difference	Medium Difference	No Difference
3	(2,2,100)	(10,50,100)	(100,100,100)
5	(2,2,2,100)	(10,50,50,100,100)	(100,100,100,100,100)
7	(2,2,2,2,100,100,100)	(10,10,50,50,100,100,100)	(100,100,100,100,100,100,100)

The start points for all test cases was the origin (0,0,...,0). The program run time, real line length, integer line length and point deviations were recorded for each test run. All four algorithms will be compared based on program time and line length. Integer and real line lengths for the algorithms was calculated within each dimension. The program time will also be evaluated across all the algorithms. The point deviations will be used to produce a maximum deviation, average deviation and sum of all deviations per test run. These three point deviation characteristics will also be compared with in each dimension. There were three test runs for each dimension for a total of nine runs per algorithm. This required a total of thirty- six runs in order to accomplish the experiment.

CONCLUSION

This chapter discussed the development of the Slope Comparison Algorithm and the error calculation program. The experimental design that was used to evaluate the SCA, MIEA, RSA and CSA approximation methods was also presented. The next chapter will discuss the results of our tests.

IV. RESULTS

Introduction

Chapter IV will discuss the results of our experiment. The MIEA, RSA, CSA, and SCA methods were tested using high, medium and low slope variations. All of the algorithms were tested in 3, 5 and 7 dimensional problem spaces. This required a run for each dimension for every slope variation. Ultimately, there were 36 test runs accomplished to cover all four algorithms. The algorithms were evaluated based on the real line length, integer line length, maximum deviation, average distance, sum of the distances and run time.

The real and integer line lengths were equal for algorithms tested using the same dimension and slope variation (See Appendix A). It was observed that all algorithms converged on the solution point and found all interior integer points (See Appendix B). The average distance between the real line and the off curve integer point, the sum of those distances, and the maximum distances (deviation) have the same variation according to each algorithm (see Table 15).

TABLE 15: SAMPLE RESULTS

3-D Med	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	112.2497	160	45.42568	22.32562	3572.099	2.483
RSA	112.2497	160	0.684291	0.447049	71.52777	5.188
CSA	112.2497	160	0.684291	0.448475	71.75592	15.032
SCA	112.2497	160	22.71284	11.92676	1908.281	1.272

The highlighted columns in Table 15 demonstrate the trend in the maximum deviation, average distance and sum of the distances. This trend allows the direct comparison of one of these criteria with the reasonable expectation that the other two will behave similarly. The maximum deviation was chosen as the primary criterion. The maximum deviation tells us the largest distance between an off-curve integer coordinate and the real line. In other words, this distance is the greatest error of all integer coordinates used to approximate the given line by a given algorithm. Computation time is the second criterion that was used for the run by run comparison of the algorithms.

High Slope Variation

The RSA and CSA methods are the most accurate algorithms for high slope. They are the same for the 3 and 5 dimensional cases. The difference in maximum slope deviation is negligible between RSA and CSA in the seventh dimension. The MIEA and SCA methods fall third and fourth in accuracy for this case. There is not much difference between these two until the 7 dimensional scenario. SCA has a maximum deviation of 31.90, while at 92.93 the MIEA has a maximum deviation that is nearly three times as great (see Table 16).

TABLE 16: HIGH SLOPE MAX DEVIATIONS

Hi Slope	3-D max dev	5-D max dev	7-D max dev
MIEA	2.827296432	3.99680384	92.93203773
RSA	0.707106781	1	1.201325026
CSA	0.707106781	1	1.12815215
SCA	2.235263492	3.60283656	31.90397524

The SCA algorithm has the fastest computation time for the high slope test runs. The times for the MIEA, CSA and SCA algorithms all show steady increases. The RSA algorithm has the third fastest computation time, but those times fluctuate between 5.158 seconds and 5.828 seconds (see Table 17).

TABLE 17: HIGH SLOPE RUN TIMES

Hi Slope	3-D run time	5-D run time	7-D run time
MIEA	2.373	3.775	5.598
RSA	5.158	5.137	5.828
CSA	14.931	24.735	36.162
SCA	0.831	0.891	3.375

The CSA algorithm has the slowest run time in all three dimension scenarios for the high slope variation case.

Medium Slope Variation

The RSA and CSA methods show the most precision in the medium slope case. The maximum deviations are the same in the third and fourth dimensions, but there is slightly greater precision in the seventh dimension for the CSA. The SCA and MIEA finish third and fourth with respect to precision (see Table 18).

TABLE 18: MED SLOPE MAX DEVIATIONS

Med Slope	3-D max dev	5-D max dev	7-D max dev
MIEA	45.42567626	77.5624668	92.93203773
RSA	0.684290851	0.95517046	1.201325026
CSA	0.684290851	0.89040928	1.12815215
SCA	22.71283813	30.9463615	31.90397524

The MIEA and SCA methods show large deviations for all three dimension scenarios. The MIEA has three times as much deviation in the seventh dimensional scenario.

The SCA method demonstrates the fastest run time for the medium slope case. The CSA method has the slowest run times (see Table 19).

TABLE 19: MED SLOPE RUN TIMES

Med Slope	3-D run time	5-D run time	7-D run time
MIEA	2.483	4.046	5.598
RSA	5.188	5.438	5.828
CSA	15.032	25.316	36.162
SCA	1.272	2.504	3.375

The RSA algorithm had the third fastest run times in all three dimension scenarios. The computation times for RSA had little variance. For dimensions 3 through 7, the times were 5.188 to 5.828 seconds. The MIEA, CSA, and SCA methods show steady processing time increases as the dimension increases.

Low Slope Variation

RSA and CSA demonstrate the greatest precision for the low slope variation case. In this case, the two algorithms have identical maximum deviations for all three dimension size scenarios. The SCA method shows greater precision for the low slope case than either of the previous slope variation cases (see Table 20).

TABLE 20: LOW SLOPE MAX DEVIATIONS

Low Slope	3-D max dev	5-D max dev	7-D max dev
MIEA	81.64965809	109.544512	130.9307341
RSA	0.816496581	1.09544512	1.309307341
CSA	0.816496581	1.09544512	1.309307341
SCA	1.414213562	3.16227766	5.291502622

In the high slope case SCA had deviations of 2.23, 3.60 and 31.90 for the corresponding dimensions (3d, 5d, 7d). In the low slope case the deviations never get higher than 5.29 in the seventh dimension. The MIEA method has its worse performance in the low slope case. The deviations are 81.64, 109.54 and 130.93 for the corresponding dimensions (3d, 5d, 7d).

Once again the SCA method proves to be the fastest algorithm (although by a small amount). The MIEA algorithm has the second fastest run time while the CSA method has the slowest time in all three dimensions (see Table 21).

TABLE 21: LOW SLOPE RUN TIMES

Low Slope	3-D run time	5-D run time	7-D run time
MIEA	2.653	4.336	5.979
RSA	5.437	5.658	5.989
CSA	15.362	25.327	35.921
SCA	2.393	4.045	5.758

The RSA had the third fastest times. Those times had little variation and ranged between 5.437 seconds and 5.979 seconds.

Algorithm Complication

The effects of algorithm complication on run time must also be considered. Typically, floating point calculations require the most computer computation time. All four of the algorithms tested require floating point calculations. "Furthermore, the rounding operations and floating-point arithmetic in procedure line DDA are still time-consuming" (Hearn and Baker, 1997:88). Floating point arithmetic occurs when division or multiplication is needed in a computation. The MIEA, RSA and CSA all perform the division of all difference variables by the largest difference (Δx, Δy, Δz....). The first pass of the algorithm requires the addition of these relative slopes until the solution value is found. For more information on the algorithms, refer to Chapter II and Chapter III. The values are then rounded and manipulated according to that algorithms logic through addition type operations. For these three algorithms there will be a one time division calculation for every dimension of the problem. For example, a 3 dimensional problem (x, y, z) will require three division operations at the start of the algorithm. The next step is rounding the fractional values that have been added together. The SCA method compares every possible slope combination. The number of slopes required is calculated based on a permutation calculation. For a seven dimensional problem a "seven choose two" permutation would be required (see Chapter III). This means a 7 dimensional problem would require 42 slope calculations for every integer step that the SCA performed in order to approximate a real valued linear function using a sequence of integer valued coordinates. Even though 42 division operations are taking place each step, the actual integer moves are merely an addition of one unit at a time. As a result there are no rounding steps in the SCA. The results previously discussed do not support any negative time effects due to floating point operations in these four algorithms. If this were the case, then the SCA would run the slowest. The CSA runs slowly due to the alpha value used to segment the real line (see chapter II). This value breaks the line into smaller segments than the RSAs slope and requires more addition steps to arrive at the same solution. It is proposed that the deliberate selection of an alpha value would improve the efficiency of the CSA approach.

Conclusion

The MIEA, CSA and SCA methods all required more run time as the dimension size of the problem increased. This did not always hold true for changes in slope for the MIEA, RSA and CSA. The SCA did have increases in time due to dimension size and reduction in slope. The RSA algorithm had the least amount of variation in its run time through out all its tests, regardless of dimension size or slope. RSA times ranged from 5.156 seconds (3d and high slope) to 5.989 seconds (7d and low slope).

The MIEA demonstrated decreasing precision as the slope varied from high to low. MIEA precision also decreased as the dimension size increased from three to seven. There is a very dramatic decrease in precision in the high slope case when the dimension size increases from five to seven. The SCA has relatively poor precision in the high and medium slope cases. In the low slope case, SCA demonstrates acceptable precision. The greatest deviation in this last case was 5.29 in seven dimensions. The RSA and CSA algorithms demonstrated the greatest precision. Their maximum deviations were nearly exact in all but three scenarios. They are as follows: 7d and high slope, 5d and medium slope, and 7d and high slope. The difference in all three of these scenarios is negligible.

The RSA method demonstrated fast and consistent run time speed as well as best precision. The run time speed is what allows it to break the tie with the CSA method. Another aspect of the RSA run time is its consistency. The RSA ran all problem scenarios regardless of dimension or slope with in 5 to 6 seconds. The precision and quickness of RSA provides us with the best overall solution.

V. CONCLUSIONS AND RECCOMMENDATIONS

INTRODUCTION

The goal of this thesis was to find the most efficient method of approximating a real valued linear function with a sequence of integer coordinates. This was accomplished through the evaluation of the MRVAT Integer Estimation Algorithm, Relative Slope Algorithm, Continuous Segment Algorithm and Slope Comparison Algorithm. These algorithms were evaluated based on six quality and error criteria: The algorithms should converge on an end point, converge on any interior integer points and the overall error between the sequence of integer coordinates and the real valued linear function (preference curve) should be characterized. The error in each individual dimension of the problem space should also be characterized. Finally, the computer processing time and the difference between the distance lengths of the sequence of integer coordinates and the real valued linear function should be determined.

The algorithms were modeled in Visual Basic 6.0. An error calculation program was used to provide the individual integer point to real line distances, the maximum distance (deviation), the average distance, and the sum of the distances. The real and integer line lengths were also provided by this subroutine.

SUMMARY OF RESULTS

It was observed that all algorithms found the interior points and converged on the solution points (See Appendix B). The four algorithms were each tested in high, medium and low slope cases. In each of these cases a test was run for three, five, and seven dimensions. This resulted in a total of 36 runs being completed. The primary criteria used for run to run analysis were the maximum deviation and the run time.

It was discovered that the MIEA algorithm was least accurate by a significant amount. This lack of precision increased with a change in slope from high to low and an increase in dimension from 3d to 7d. The SCA method also proved to be relatively imprecise. The precision was acceptable in the low slope cases, but there were large deviations in the 7d and high slope scenario and in all dimension scenarios for the medium slope case. The CSA and RSA slope were found to be the most precise of the algorithms. The largest deviation for either of the algorithms was 1.309.

The CSA algorithm proved to be extremely slow in all slope cases and dimension scenarios compared to the other three algorithms. The SCA had the quickest run time of all algorithms evaluated during this thesis. The MIEA required 5.97 seconds for its longest test run for a 7d/ low slope problem. The RSA algorithm had run times that ranged between 5.148 seconds and 5.989 seconds. RSA ran the third in computation speed and consistently solved the approximation problem in 5 to 6 seconds.

RECOMMENDATIONS

Based on the results of this thesis, it is recommended that the Relative Slope Algorithm be utilized in the Mission Resource Value Assessment Tool program. This is the next step in testing the effectiveness of the RSA and judging its impact on the MRVAT. It is also recommended that the RSA be expanded to a 20 dimensional problem. A problem of this size would further demonstrate the applicability of an improved MRVAT to the Advanced Logistics Project.

Appendix A: Algorithm Test Results

Three Dimensional Case:

End	2	2	100
Start	0	0	0

3-D Hi	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	100.04	104	2.827296	1.423203	148.0131	2.373
RSA	100.04	104	0.707107	0.36701	38.16907	5.158
CSA	100.04	104	0.707107	0.36701	38.16907	14.931
SCA	100.04	104	2.235263	1.185098	123.2502	0.831

End	10	50	100
Start	0	0	0

3-D Med	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	112.2497	160	45.42568	22.32562	3572.099	2.483
RSA	112.2497	160	0.684291	0.447049	71.52777	5.188
CSA	112.2497	160	0.684291	0.448475	71.75592	15.032
SCA	112.2497	160	22.71284	11.92676	1908.281	1.272

End	100	100	100
Start	0	0	0

3-D Low	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	173.2051	300	81.64966	52.03772	15611.32	2.653
RSA	173.2051	300	0.816497	0.544331	163.2993	5.437
CSA	173.2051	300	0.816497	0.544331	163.2993	15.362
SCA	173.2051	300	1.414214	1.011022	303.3065	2.393

Five Dimensional Case:

End	2	2	2	2	100
Start	0	0	0	0	0

5-D Hi	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	100.08	108	3.996804	2.036701	219.9638	3.775
RSA	100.08	108	1	0.536645	57.95765	5.137
CSA	100.08	108	1	0.536645	57.95765	24.735
SCA	100.08	108	3.602837	1.878339	202.8606	0.891

End	10	50	50	100	100
Start	0	0	0	0	0

5-D Med	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	158.4298	310	77.56247	45.89123	14226.28	4.046
RSA	158.4298	310	0.95517	0.676949	209.8541	5.438
CSA	158.4298	310	0.890409	0.678624	210.3735	25.316
SCA	158.4298	310	30.94636	15.448	4788.879	2.504

End	100	100	100	100	100
Start	0	0	0	0	0

5-D Low	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	223.6068	500	109.5445	75.67659	37838.3	4.336
RSA	223.6068	500	1.095445	0.795949	397.9745	5.658
CSA	223.6068	500	1.095445	0.795949	397.9745	25.327
SCA	223.6068	500	3.162278	2.552431	1276.215	4.045

Seven Dimensional Case:

End	2	2	2	2	100	100	100
Start	0	0	0	0	0	0	0

7-D Hi	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	173.2513	308	81.69317	50.87788	15670.39	5.428
RSA	173.2513	308	1.280492	0.830661	255.8436	5.397
CSA	173.2513	308	1.280492	0.830661	255.8436	35.411
SCA	173.2513	308	3.286088	2.236041	688.7007	2.564

End	10	10	50	50	100	100	100
Start	0	0	0	0	0	0	0

7-D Med	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	187.6166	420	92.93204	57.22633	24035.06	5.598
RSA	187.6166	420	1.201325	0.827896	347.7164	5.828
CSA	187.6166	420	1.128152	0.829251	348.2854	36.162
SCA	187.6166	420	31.90398	16.02219	6729.32	3.375

End	100	100	100	100	100	100	100
Start	0	0	0	0	0	0	0

7-D Low	Real_L	Integer_L	Max dev.	Avg. Dist.	Sum Dist.	Run Time
MIEA	264.5751	700	130.9307	93.72438	65607.07	5.979
RSA	264.5751	700	1.309307	0.980102	686.0712	5.989
CSA	264.5751	700	1.309307	0.980102	686.0712	35.921
SCA	264.5751	700	5.291503	4.478259	3134.781	5.758

Appendix B: Sample Output Examples (3d)

End: (5, 10, 15)
Start: (0, 0, 0)

MIEA Sample Output:

Integer Points

Iterations	*x*	*y*	*z*	*t*	*u*	*v*	*w*
1			1	0	0	0	0
2			2	0	0	0	0
3			3	0	0	0	0
4			4	0	0	0	0
5			5	0	0	0	0
6			6	0	0	0	0
7			7	0	0	0	0
8			8	0	0	0	0
9			9	0	0	0	0
10			10	0	0	0	0
11			11	0	0	0	0
12			12	0	0	0	0
13			13	0	0	0	0
14			14	0	0	0	0
15			15	0	0	0	0
16		1	15	0	0	0	0
17		2	15	0	0	0	0
18		3	15	0	0	0	0
19		4	15	0	0	0	0
20		5	15	0	0	0	0
21		6	15	0	0	0	0
22		7	15	0	0	0	0
23		8	15	0	0	0	0
24		9	15	0	0	0	0
25		10	15	0	0	0	0
26	1	10	15	0	0	0	0
27	2	10	15	0	0	0	0
28	*3*	*10*	*15*	*0*	*0*	*0*	*0*
29	4	10	15	0	0	0	0
30	5	10	15	0	0	0	0

Real Line Points

Iterations	*PoCurve1*	*PoCurve2*	*PoCurve3*
1	0.214286	0.428571	0.642857
2	0.428571	0.857143	1.285714
3	0.642857	1.285714	1.928571
4	0.857143	1.714286	2.571429
5	1.071429	2.142857	3.214286
6	1.285714	2.571429	3.857143
7	1.5	3	4.5
8	1.714286	3.428571	5.142857
9	1.928571	3.857143	5.785714
10	*2.14286*	*4.28571*	*6.42857*
11	2.357143	4.714286	7.071429
12	2.571429	5.142857	7.714286
13	2.785714	5.571429	8.357143
14	**3**	**6**	**9**
15	3.214286	6.428571	9.642857
16	3.357143	6.714286	10.07143
17	3.5	7	10.5
18	3.642857	7.285714	10.92857
19	3.785714	7.571429	11.35714
20	3.928571	7.857143	11.78571
21	4.071429	8.142857	12.21429
22	4.214286	8.428571	12.64286
23	4.357143	8.714286	13.07143
24	4.5	9	13.5
25	4.642857	9.285714	13.92857
26	4.714286	9.428571	14.14286
27	4.785714	9.571429	14.35714
28	*4.85714*	*9.71429*	*14.5714*
29	4.928571	9.857143	14.78571
30	**5**	**10**	**15**

RSA Sample Output:

Integer Points

Iteration	*x*	*y*	*z*	*t*	*u*	*v*	*w*
1	0	0	1	0	0	0	0
2	0	1	1	0	0	0	0
3	0	1	2	0	0	0	0
4	1	1	2	0	0	0	0
5	1	1	3	0	0	0	0
6	1	2	3	0	0	0	0
7	1	2	4	0	0	0	0
8	1	3	4	0	0	0	0
9	1	3	5	0	0	0	0
10	*2*	*3*	*5*	*0*	*0*	*0*	*0*
11	2	3	6	0	0	0	0
12	2	4	6	0	0	0	0
13	2	4	7	0	0	0	0
14	2	5	7	0	0	0	0
15	2	5	8	0	0	0	0
16	3	5	8	0	0	0	0
17	3	5	9	0	0	0	0
18	3	6	9	0	0	0	0
19	3	6	10	0	0	0	0
20	3	7	10	0	0	0	0
21	3	7	11	0	0	0	0
22	4	7	11	0	0	0	0
23	4	7	12	0	0	0	0
24	4	8	12	0	0	0	0
25	4	8	13	0	0	0	0
26	4	9	13	0	0	0	0
27	4	9	14	0	0	0	0
28	*5*	*9*	*14*	*0*	*0*	*0*	*0*
29	5	9	15	0	0	0	0
30	5	10	15	0	0	0	0

Real Line Points

Iteration	PoCurve1	PoCurve2	PoCurve3
1	0.214285714	0.428571429	0.642857143
2	0.357142857	0.714285714	1.071428571
3	0.571428571	1.142857143	1.714285714
4	0.642857143	1.285714286	1.928571429
5	0.857142857	1.714285714	2.571428571
6	**1**	**2**	**3**
7	1.214285714	2.428571429	3.642857143
8	1.357142857	2.714285714	4.071428571
9	1.571428571	3.142857143	4.714285714
10	*1.642857143*	*3.285714286*	*4.928571429*
11	1.857142857	3.714285714	5.571428571
12	**2**	**4**	**6**
13	2.214285714	4.428571429	6.642857143
14	2.357142857	4.714285714	7.071428571
15	2.571428571	5.142857143	7.714285714
16	2.642857143	5.285714286	7.928571429
17	2.857142857	5.714285714	8.571428571
18	**3**	**6**	**9**
19	3.214285714	6.428571429	9.642857143
20	3.357142857	6.714285714	10.07142857
21	3.571428571	7.142857143	10.71428571
22	3.642857143	7.285714286	10.92857143
23	3.857142857	7.714285714	11.57142857
24	**4**	**8**	**12**
25	4.214285714	8.428571429	12.64285714
26	4.357142857	8.714285714	13.07142857
27	4.571428571	9.142857143	13.71428571
28	*4.642857143*	*9.285714286*	*13.92857143*
29	4.857142857	9.714285714	14.57142857
30	**5**	**10**	**15**

CSA Sample Output:

Integer Points

Iteration	*x*	*y*	*z*	*t*	*u*	*v*	*w*
1	0	0	1	0	0	0	0
2	0	1	1	0	0	0	0
3	0	1	2	0	0	0	0
4	1	1	2	0	0	0	0
5	1	2	2	0	0	0	0
6	1	2	3	0	0	0	0
7	1	2	4	0	0	0	0
8	1	3	4	0	0	0	0
9	1	3	5	0	0	0	0
10	*2*	*3*	*5*	*0*	*0*	*0*	*0*
11	2	4	5	0	0	0	0
12	2	4	6	0	0	0	0
13	2	4	7	0	0	0	0
14	2	5	7	0	0	0	0
15	2	5	8	0	0	0	0
16	3	5	8	0	0	0	0
17	3	6	8	0	0	0	0
18	3	6	9	0	0	0	0
19	3	6	10	0	0	0	0
20	3	7	10	0	0	0	0
21	3	7	11	0	0	0	0
22	4	7	11	0	0	0	0
23	4	8	11	0	0	0	0
24	4	8	12	0	0	0	0
25	4	8	13	0	0	0	0
26	4	9	13	0	0	0	0
27	4	9	14	0	0	0	0
28	*5*	*9*	*14*	*0*	*0*	*0*	*0*
29	5	10	14	0	0	0	0
30	5	10	15	0	0	0	0

Real Line Points

Iteration	PoCurve1	PoCurve2	PoCurve3
1	0.214285714	0.428571429	0.642857143
2	0.357142857	0.714285714	1.071428571
3	0.571428571	1.142857143	1.714285714
4	0.642857143	1.285714286	1.928571429
5	0.785714286	1.571428571	2.357142857
6	**1**	**2**	**3**
7	1.214285714	2.428571429	3.642857143
8	1.357142857	2.714285714	4.071428571
9	1.571428571	3.142857143	4.714285714
10	*1.642857143*	*3.285714286*	*4.928571429*
11	1.785714286	3.571428571	5.357142857
12	**2**	**4**	**6**
13	2.214285714	4.428571429	6.642857143
14	2.357142857	4.714285714	7.071428571
15	2.571428571	5.142857143	7.714285714
16	2.642857143	5.285714286	7.928571429
17	2.785714286	5.571428571	8.357142857
18	**3**	**6**	**9**
19	3.214285714	6.428571429	9.642857143
20	3.357142857	6.714285714	10.07142857
21	3.571428571	7.142857143	10.71428571
22	3.642857143	7.285714286	10.92857143
23	3.785714286	7.571428571	11.35714286
24	**4**	**8**	**12**
25	4.214285714	8.428571429	12.64285714
26	4.357142857	8.714285714	13.07142857
27	4.571428571	9.142857143	13.71428571
28	*4.642857143*	*9.285714286*	*13.92857143*
29	4.785714286	9.571428571	14.35714286
30	**5**	**10**	**15**

SCA Sample Output:

Integer Points

Iterations	x	y	z	t	u	v	w
1	0	0	1	0	0	0	0
2	0	0	2	0	0	0	0
3	0	0	3	0	0	0	0
4	0	0	4	0	0	0	0
5	0	0	5	0	0	0	0
6	0	1	5	0	0	0	0
7	0	1	6	0	0	0	0
8	0	2	6	0	0	0	0
9	0	2	7	0	0	0	0
10	0	3	7	0	0	0	0
11	0	3	8	0	0	0	0
12	0	4	8	0	0	0	0
13	0	4	9	0	0	0	0
14	0	5	9	0	0	0	0
15	0	6	9	0	0	0	0
16	0	6	10	0	0	0	0
17	1	6	10	0	0	0	0
18	1	7	10	0	0	0	0
19	1	7	11	0	0	0	0
20	2	7	11	0	0	0	0
21	2	8	11	0	0	0	0
22	2	8	12	0	0	0	0
23	*3*	*8*	*12*	*0*	*0*	*0*	*0*
24	3	9	12	0	0	0	0
25	3	9	13	0	0	0	0
26	4	9	13	0	0	0	0
27	4	10	13	0	0	0	0
28	4	10	14	0	0	0	0
29	5	10	14	0	0	0	0
30	5	10	15	0	0	0	0

Real Line Points

Iterations	***PoCurve1***	***PoCurve2***	***PoCurve3***
1	0.214286	0.428571	0.642857
2	0.428571	0.857143	1.285714
3	0.642857	1.285714	1.928571
4	0.857143	1.714286	2.571429
5	1.071429	2.142857	3.214286
6	1.214286	2.428571	3.642857
7	1.428571	2.857143	4.285714
8	1.571429	3.142857	4.714286
9	1.785714	3.571429	5.357143
10	1.928571	3.857143	5.785714
11	2.142857	4.285714	6.428571
12	2.285714	4.571429	6.857143
13	2.5	5	7.5
14	2.642857	5.285714	7.928571
15	2.785714	5.571429	8.357143
16	**3**	**6**	**9**
17	3.071429	6.142857	9.214286
18	3.214286	6.428571	9.642857
19	3.428571	6.857143	10.28571
20	3.5	7	10.5
21	3.642857	7.285714	10.92857
22	3.857143	7.714286	11.57143
23	*3.92857*	*7.85714*	*11.7857*
24	4.071429	8.142857	12.21429
25	4.285714	8.571429	12.85714
26	4.357143	8.714286	13.07143
27	4.5	9	13.5
28	4.714286	9.428571	14.14286
29	4.785714	9.571429	14.35714
30	**5**	**10**	**15**

Appendix C: MIEA SOURCE CODE

```
Sub Mslope()
'
' Mslope Macro
' Macro recorded 4/13/2001 by rantoine
'
Dim ValueX As Double
Dim ValueY As Double
Dim ValueZ As Double
Dim ValueT As Double
Dim ValueU As Double
Dim ValueV As Double
Dim ValueW As Double

Dim X2 As Integer
Dim Y2 As Integer
Dim Z2 As Integer
Dim T2 As Integer
Dim U2 As Integer
Dim V2 As Integer
Dim W2 As Integer

Dim i As Integer
Dim n As Integer
Dim z As Integer

Dim num As Integer
Dim Denominator As Integer
Dim Biggest As Integer
Dim Increment As Double

Dim Xincrement As Double
Dim Yincrement As Double
Dim Zincrement As Double
Dim Tincrement As Double
Dim Uincrement As Double
Dim Vincrement As Double
Dim Wincrement As Double

Dim SortX As Double
Dim SortY As Double
Dim SortZ As Double
Dim SortT As Double
Dim SortU As Double
Dim SortV As Double
Dim SortW As Double


'Get initial starting values
ValueX = Cells(2, 2).Value
ValueY = Cells(3, 2).Value
```

```
ValueZ = Cells(4, 2).Value
ValueT = Cells(5, 2).Value
ValueU = Cells(6, 2).Value
ValueV = Cells(7, 2).Value
ValueW = Cells(8, 2).Value

'Get final values
X2 = Cells(2, 3).Value
Y2 = Cells(3, 3).Value
Z2 = Cells(4, 3).Value
T2 = Cells(5, 3).Value
U2 = Cells(6, 3).Value
V2 = Cells(7, 3).Value
W2 = Cells(8, 3).Value

z = 0
num = 0

For Biggest = 1 To 7
Denominator = Cells(10 + z, 2).Value
If Denominator > num Then
   num = Denominator
   Cells(18, 2) = num
End If

z = z + 1

Next Biggest


'Get Slope Values for sorting.
SortX = Cells(21, 2).Value
SortY = Cells(22, 2).Value
SortZ = Cells(23, 2).Value
SortT = Cells(24, 2).Value
SortU = Cells(25, 2).Value
SortV = Cells(26, 2).Value
SortW = Cells(27, 2).Value

'Send slope values to sorter.
Cells(21, 3) = SortX
Cells(22, 3) = SortY
Cells(23, 3) = SortZ
Cells(24, 3) = SortT
Cells(25, 3) = SortU
Cells(26, 3) = SortV
Cells(27, 3) = SortW



'
End Sub
```

```
Sub Msort()
```

```
'
' Msort Macro
' Macro recorded 4/13/2001 by rantoine
'

'
    Range("C21:C27").Select
    Selection.SORT Key1:=Range("C21"), Order1:=xlDescending, Header:=xlGuess _
        , OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
End Sub
```

```
Sub Madd()
'
' Madd Macro
' Macro recorded 4/13/2001 by rantoine
'
'
Dim ValueX As Double
Dim ValueY As Double
Dim ValueZ As Double
Dim ValueT As Double
Dim ValueU As Double
Dim ValueV As Double
Dim ValueW As Double

Dim X2 As Integer
Dim Y2 As Integer
Dim Z2 As Integer
Dim T2 As Integer
Dim U2 As Integer
Dim V2 As Integer
Dim W2 As Integer

Dim i As Integer
Dim n As Integer
Dim z As Integer
Dim b As Integer

Dim num As Integer
Dim Denominator As Integer
Dim Biggest As Integer
Dim Increment As Double
Dim Squeeze As Integer

Dim Xincrement As Double
Dim Yincrement As Double
Dim Zincrement As Double
Dim Tincrement As Double
Dim Uincrement As Double
Dim Vincrement As Double
Dim Wincrement As Double

Dim SortX As Double
Dim SortY As Double
Dim SortZ As Double
Dim SortT As Double
Dim SortU As Double
Dim SortV As Double
Dim SortW As Double

Dim Xround1 As Integer
Dim Yround1 As Integer
Dim Zround1 As Integer
Dim Tround1 As Integer
Dim Uround1 As Integer
Dim Vround1 As Integer
Dim Wround1 As Integer

Dim Xround2 As Integer
Dim Yround2 As Integer
Dim Zround2 As Integer
Dim Tround2 As Integer
Dim Uround2 As Integer
Dim Vround2 As Integer
Dim Wround2 As Integer

Dim a As Integer
Dim c As Integer
Dim d As Integer
Dim e As Integer
Dim f As Integer
Dim g As Integer
Dim h As Integer
Dim j As Integer
Dim k As Integer
Dim l As Integer
Dim m As Integer

Dim Xholder As Integer
Dim Yholder As Integer
Dim Zholder As Integer
Dim Tholder As Integer
Dim Uholder As Integer
Dim Vholder As Integer
Dim Wholder As Integer


'Get initial starting values
ValueX = Cells(2, 2).Value
ValueY = Cells(3, 2).Value
ValueZ = Cells(4, 2).Value
ValueT = Cells(5, 2).Value
ValueU = Cells(6, 2).Value
ValueV = Cells(7, 2).Value
ValueW = Cells(8, 2).Value

'Get final values
X2 = Cells(2, 3).Value
```

```
Y2 = Cells(3, 3).Value
Z2 = Cells(4, 3).Value
T2 = Cells(5, 3).Value
U2 = Cells(6, 3).Value
V2 = Cells(7, 3).Value
W2 = Cells(8, 3).Value


i = 0
n = 0

Do

Xincrement = ValueX
Yincrement = ValueY
Zincrement = ValueZ
Tincrement = ValueT
Uincrement = ValueU
Vincrement = ValueV
Wincrement = ValueW

'Now increment values according to slope ratios
If Cells(21, 2) = Cells(21 + n, 3) And ValueX < X2 Then
  Do
    ValueX = ValueX + Cells(21, 2)
    Cells(32 + i, 2) = ValueX
    i = i + 1
  Loop While (ValueX < X2)
End If

If Cells(22, 2) = Cells(21 + n, 3) And ValueY < Y2 Then
  Do
    ValueY = ValueY + Cells(22, 2)
    Cells(32 + i, 3) = ValueY
    i = i + 1
  Loop While (ValueY < Y2)
End If

If Cells(23, 2) = Cells(21 + n, 3) And ValueZ < Z2 Then
  Do
    ValueZ = ValueZ + Cells(23, 2)
    Cells(32 + i, 4) = ValueZ
    i = i + 1
  Loop While (ValueZ < Z2)
End If

If Cells(24, 2) = Cells(21 + n, 3) And ValueT < T2 Then
  Do
    ValueT = ValueT + Cells(24, 2)
    Cells(32 + i, 5) = ValueT
    i = i + 1
  Loop While (ValueT < T2)
End If
```

```
If Cells(25, 2) = Cells(21 + n, 3) And ValueU < U2 Then
   Do
      ValueU = ValueU + Cells(25, 2)
      Cells(32 + i, 6) = ValueU
      i = i + 1
   Loop While (ValueU < U2)
End If

If Cells(26, 2) = Cells(21 + n, 3) And ValueV < V2 Then
   Do
      ValueV = ValueV + Cells(26, 2)
      Cells(32 + i, 7) = ValueV
      i = i + 1
   Loop While (ValueV < V2)
End If

If Cells(27, 2) = Cells(21 + n, 3) And ValueW < W2 Then
   Do
      ValueW = ValueW + Cells(27, 2)
      Cells(32 + i, 8) = ValueW
      i = i + 1
   Loop While (ValueW < W2)
End If


n = n + 1

Loop While ((ValueX < X2) Or (ValueY < Y2) Or (ValueZ < Z2) Or (ValueT < T2) Or (ValueU < U2) Or
(ValueV < V2) Or (ValueW < W2))

Cells(2, 11) = i
'
'b = 0
'Now Squeeze or fill down the rounding Table.

For Squeeze = 1 To i

   Xround1 = Cells(32 + b, 11)
   Yround1 = Cells(32 + b, 12)
   Zround1 = Cells(32 + b, 13)
   Tround1 = Cells(32 + b, 14)
   Uround1 = Cells(32 + b, 15)
   Vround1 = Cells(32 + b, 16)
   Wround1 = Cells(32 + b, 17)

   Xround2 = Cells(33 + b, 11)
   Yround2 = Cells(33 + b, 12)
   Zround2 = Cells(33 + b, 13)
   Tround2 = Cells(33 + b, 14)
   Uround2 = Cells(33 + b, 15)
   Vround2 = Cells(33 + b, 16)
   Wround2 = Cells(33 + b, 17)

   If Xround1 < Xround2 Or Xround1 > Xround2 Then
```

```
  Cells(32 + b, 20) = Xround1
Else
  If (Xround1 - Xround2) = 0 And Xround1 > 0 And Xround2 > 0 Then
    b = b
  End If
End If

If Yround1 < Yround2 Or Yround1 > Yround2 Then
  Cells(32 + b, 21) = Yround1
Else
  If (Yround1 - Yround2) = 0 And Yround1 > 0 And Yround2 > 0 Then
    b = b
  End If
End If

If Zround1 < Zround2 Or Zround1 > Zround2 Then
  Cells(32 + b, 22) = Zround1
Else
  If (Zround1 - Zround2) = 0 And Zround1 > 0 And Zround2 > 0 Then
    b = b
  End If
End If

If Tround1 < Tround2 Or Tround1 > Tround2 Then
  Cells(32 + b, 23) = Tround1
Else
  If (Tround1 - Tround2) = 0 And Tround1 > 0 And Tround2 > 0 Then
    b = b
  End If
End If

If Uround1 < Uround2 Or Uround1 > Uround2 Then
  Cells(32 + b, 24) = Uround1
Else
  If (Uround1 - Uround2) = 0 And Uround1 > 0 And Uround2 > 0 Then
    b = b
  End If
End If

If Vround1 < Vround2 Or Vround1 > Vround2 Then
  Cells(32 + b, 25) = Vround1
Else
  If (Vround1 - Vround2) = 0 And Vround1 > 0 And Vround2 > 0 Then
    b = b
  End If
End If

If Wround1 < Wround2 Or Wround1 > Wround2 Then
  Cells(32 + b, 26) = Wround1
Else
  If (Wround1 - Wround2) = 0 And Wround1 > 0 And Wround2 > 0 Then
    b = b
  End If
End If
```

```
 b = b + 1
Next Squeeze

'Second pass for fill down.  This part searches the first fill down and writes the
'values so there are no spaces in between the coordinate values.

a = 0
c = 0

Do
   If Cells(32 + a, 20) > 0 Then
      Xholder = Cells(32 + a, 20)
      Cells(32 + c, 29) = Xholder
      c = c + 1
   Else
      c = c
   End If

   a = a + 1
Loop While (Xholder < X2)

a = 0
c = 0

Do
   If Cells(32 + a, 21) > 0 Then
      Yholder = Cells(32 + a, 21)
      Cells(32 + c, 30) = Yholder
      c = c + 1
   Else
      c = c
   End If

   a = a + 1
Loop While (Yholder < Y2)

a = 0
c = 0

Do
   If Cells(32 + a, 22) > 0 Then
      Zholder = Cells(32 + a, 22)
      Cells(32 + c, 31) = Zholder
      c = c + 1
   Else
      c = c
   End If

   a = a + 1
Loop While (Zholder < Z2)

a = 0
c = 0
```

```
Do
   If Cells(32 + a, 23) > 0 Then
      Tholder = Cells(32 + a, 23)
      Cells(32 + c, 32) = Tholder
      c = c + 1
   Else
      c = c
   End If

   a = a + 1
Loop While (Tholder < T2)

a = 0
c = 0

Do
   If Cells(32 + a, 24) > 0 Then
      Uholder = Cells(32 + a, 24)
      Cells(32 + c, 33) = Uholder
      c = c + 1
   Else
      c = c
   End If

   a = a + 1
Loop While (Uholder < U2)

a = 0
c = 0

Do
   If Cells(32 + a, 25) > 0 Then
      Vholder = Cells(32 + a, 25)
      Cells(32 + c, 34) = Vholder
      c = c + 1
   Else
      c = c
   End If

   a = a + 1
Loop While (Vholder < V2)

a = 0
c = 0

Do
   If Cells(32 + a, 26) > 0 Then
      Wholder = Cells(32 + a, 26)
      Cells(32 + c, 35) = Wholder
      c = c + 1
   Else
      c = c
   End If
```

```
 a = a + 1
Loop While (Wholder < W2)

'3rd Fill Down, now we are taking each coordinate
'set and setting them end to end.

Dim xplace As Integer
Dim yplace As Integer
Dim zplace As Integer
Dim tplace As Integer
Dim uplace As Integer
Dim vplace As Integer
Dim wplace As Integer

d = 0
e = 0
f = 0
g = 0
h = 0
j = 0
k = 0
l = 0
m = 0

xplace = Cells(32 + d, 29).Value
yplace = Cells(32 + d, 30).Value
zplace = Cells(32 + d, 31).Value
tplace = Cells(32 + d, 32).Value
uplace = Cells(32 + d, 33).Value
vplace = Cells(32 + d, 34).Value
wplace = Cells(32 + d, 35).Value

Do

If Cells(21, 2).Value = Cells(21 + e, 3).Value And xplace < X2 Then
  Do
    xplace = Cells(32 + d, 29).Value
    If xplace > 0 Then
      Cells(32 + f, 39).Value = xplace
      f = f + 1
      d = d + 1
    Else
      d = d
    End If
  Loop While (xplace < X2)
End If


If Cells(22, 2).Value = Cells(21 + e, 3).Value And yplace < Y2 Then
  Do
    yplace = Cells(32 + g, 30).Value
    If yplace > 0 Then
      Cells(32 + f, 40).Value = yplace
```

```
            f = f + 1
            g = g + 1
        Else
            g = g
        End If
    Loop While (yplace < Y2)
End If


If Cells(23, 2).Value = Cells(21 + e, 3).Value And zplace < Z2 Then
    Do
        zplace = Cells(32 + h, 31).Value
        If zplace > 0 Then
            Cells(32 + f, 41).Value = zplace
            f = f + 1
            h = h + 1
        Else
            h = h
        End If
    Loop While (zplace < Z2)
End If


If Cells(24, 2).Value = Cells(21 + e, 3).Value And tplace < T2 Then
    Do
        tplace = Cells(32 + j, 32).Value
        If tplace > 0 Then
            Cells(32 + f, 42).Value = tplace
            f = f + 1
            j = j + 1
        Else
            j = j
        End If
    Loop While (tplace < T2)
End If


If Cells(25, 2).Value = Cells(21 + e, 3).Value And uplace < U2 Then
    Do
        uplace = Cells(32 + k, 33).Value
        If uplace > 0 Then
            Cells(32 + f, 43).Value = uplace
            f = f + 1
            k = k + 1
        Else
            k = k
        End If
    Loop While (uplace < U2)
End If


If Cells(26, 2).Value = Cells(21 + e, 3).Value And vplace < V2 Then
    Do
        vplace = Cells(32 + l, 34).Value
        If vplace > 0 Then
            Cells(32 + f, 44).Value = vplace
            f = f + 1
            l = l + 1
        Else
            l = l
        End If
    Loop While (vplace < V2)
End If


If Cells(27, 2).Value = Cells(21 + e, 3).Value And wplace < W2 Then
    Do
        wplace = Cells(32 + m, 35).Value
        If wplace > 0 Then
            Cells(32 + f, 45).Value = wplace
            f = f + 1
            m = m + 1
        Else
            m = m
        End If
    Loop While (wplace < W2)
End If


e = e + 1

Loop While ((xplace < X2) Or (yplace < Y2) Or (zplace < Z2) Or (tplace < T2) Or (uplace < U2) Or
(vplace < V2) Or (wplace < W2))


Cells(29, 47) = f
'Cells(125, 2) = f

End Sub
```

```
Sub FinalFill()
'
' FinalFill Macro
' Macro recorded 4/24/2001 by rantoine
'
Dim xplace As Integer
Dim yplace As Integer
Dim zplace As Integer
Dim tplace As Integer
Dim uplace As Integer
Dim vplace As Integer
Dim wplace As Integer

Dim X2 As Integer
Dim Y2 As Integer
Dim Z2 As Integer
Dim T2 As Integer
Dim U2 As Integer
Dim V2 As Integer
Dim W2 As Integer

Dim xpart As Integer
Dim ypart As Integer
Dim zpart As Integer
Dim tpart As Integer
Dim upart As Integer
Dim vpart As Integer
Dim wart As Integer

Dim a As Integer
Dim b As Integer
Dim c As Integer
Dim d As Integer
Dim e As Integer
Dim f As Integer
Dim g As Integer
Dim h As Integer
Dim i As Integer
Dim j As Integer
Dim k As Integer
Dim l As Integer
Dim m As Integer
Dim n As Integer

'Get final values
X2 = Cells(2, 3).Value
Y2 = Cells(3, 3).Value
Z2 = Cells(4, 3).Value
T2 = Cells(5, 3).Value
U2 = Cells(6, 3).Value
V2 = Cells(7, 3).Value
W2 = Cells(8, 3).Value

FFill = Cells(29, 47)


'Go variable by variable and fill from last number in
'column until the final(lowest in column) of any variable.
a = 0
b = 0

Do
xplace = Cells(32 + a, 39)

If xplace = X2 Then
   For xpart = 1 To (FFill - a)
    Cells(32 + a + b, 39) = xplace
    b = b + 1
   Next xpart
Else
   x = x
End If
```

```
a = a + 1
Loop While (xplace < X2)


c = 0
d = 0

Do
yplace = Cells(32 + c, 40)

If yplace = Y2 Then
   For ypart = 1 To (FFill - c)
    Cells(32 + c + d, 40) = yplace
    d = d + 1
   Next ypart
Else
   x = x
End If

c = c + 1
Loop While (yplace < Y2)

e = 0
f = 0

Do
zplace = Cells(32 + e, 41)

If zplace = Z2 Then
   For zpart = 1 To (FFill - e)
    Cells(32 + e + f, 41) = zplace
    f = f + 1
   Next zpart
Else
   x = x
End If

e = e + 1
Loop While (zplace < Z2)

g = 0
h = 0

Do
tplace = Cells(32 + g, 42)

If tplace = T2 Then
   For tpart = 1 To (FFill - g)
    Cells(32 + g + h, 42) = tplace
    h = h + 1
   Next tpart
Else
   x = x
```

```
End If

g = g + 1
Loop While (tplace < T2)

i = 0
j = 0

Do
uplace = Cells(32 + i, 43)

If uplace = U2 Then
   For upart = 1 To (FFill - i)
    Cells(32 + i + j, 43) = uplace
    j = j + 1
   Next upart
Else
   x = x
End If

i = i + 1
Loop While (uplace < U2)

k = 0
l = 0

Do
vplace = Cells(32 + k, 44)

If vplace = V2 Then
   For vpart = 1 To (FFill - k)
    Cells(32 + k + l, 44) = vplace
    l = l + 1
   Next vpart
Else
   x = x
End If

k = k + 1
Loop While (vplace < V2)

m = 0
n = 0

Do
wplace = Cells(32 + m, 45)

If wplace = W2 Then
   For wpart = 1 To (FFill - m)
    Cells(32 + m + n, 45) = wplace
    n = n + 1
   Next wpart
Else
   x = x
```

```
End If

m = m + 1
Loop While (wplace < W2)



'
End Sub
```

```
Sub BuildSlopeAndError()
'
' BuildSlopeAndError Macro
' Macro recorded 4/24/2001 by rantoine
'

'
Dim Start As Double
Dim Finish As Double
Dim TotalTime As Double
Dim CountProgram As Integer

n = 0

Start = Timer

For CountProgram = 1 To 1
   Application.Run "MIEArow.xls!Mslope"
   Application.Run "MIEArow.xls!Msort"
   Application.Run "MIEArow.xls!Madd"
   Application.Run "MIEArow.xls!FinalFill"
   n = n + 1

Next CountProgram

'Calculate Elapsed Time
   Finish = Timer   ' Set end time.
   TotalTime = Finish - Start    ' Calculate total time.

'Write program time to excel
Cells(29, 4) = TotalTime
Cells(32, 75) = TotalTime
MsgBox "Program time is  " & TotalTime & " seconds"
   Application.Run "MIEArow.xls!RSAError"

End Sub
```

Appendix D: RSA SOURCE CODE

```
Sub RelSlope()
'
' RelSlope Macro
' Macro recorded 4/9/2001 by rantoine
'
Dim ValueX As Double
Dim ValueY As Double
Dim ValueZ As Double
Dim ValueT As Double
Dim ValueU As Double
Dim ValueV As Double
Dim ValueW As Double

Dim X2 As Integer
Dim Y2 As Integer
Dim Z2 As Integer
Dim T2 As Integer
Dim U2 As Integer
Dim V2 As Integer
Dim W2 As Integer

Dim i As Integer
Dim n As Integer
Dim z As Integer

Dim num As Integer
Dim Denominator As Integer
Dim Biggest As Integer
Dim Increment As Double

Dim Xincrement As Double
Dim Yincrement As Double
Dim Zincrement As Double
Dim Tincrement As Double
Dim Uincrement As Double
Dim Vincrement As Double
Dim Wincrement As Double

Dim Sortx As Double
Dim Sorty As Double
Dim Sortz As Double
Dim Sortt As Double
Dim Sortu As Double
Dim Sortv As Double
Dim Sortw As Double

'Get initial starting values
ValueX = Cells(2, 2).Value
ValueY = Cells(3, 2).Value
ValueZ = Cells(4, 2).Value
ValueT = Cells(5, 2).Value
```

```
ValueU = Cells(6, 2).Value
ValueV = Cells(7, 2).Value
ValueW = Cells(8, 2).Value

'Get final values
X2 = Cells(2, 3).Value
Y2 = Cells(3, 3).Value
Z2 = Cells(4, 3).Value
T2 = Cells(5, 3).Value
U2 = Cells(6, 3).Value
V2 = Cells(7, 3).Value
W2 = Cells(8, 3).Value

z = 0
num = 0

For Biggest = 1 To 7
Denominator = Cells(10 + z, 2).Value
If Denominator > num Then
   num = Denominator
   Cells(18, 2) = num
End If

z = z + 1

Next Biggest

i = 0
n = 0

Do

Xincrement = ValueX
Yincrement = ValueY
Zincrement = ValueZ
Tincrement = ValueT
Uincrement = ValueU
Vincrement = ValueV
Wincrement = ValueW

'Now increment values according to slope ratios
If ValueX < Cells(2, 3).Value Then
   ValueX = Xincrement + Cells(21, 2)
End If
If ValueY < Cells(3, 3).Value Then
   ValueY = Yincrement + Cells(22, 2)
End If
If ValueZ < Cells(4, 3).Value Then
   ValueZ = Zincrement + Cells(23, 2)
End If
If ValueT < Cells(5, 3).Value Then
   ValueT = Tincrement + Cells(24, 2)
End If
If ValueU < Cells(6, 3).Value Then
```

```
    ValueU = Uincrement + Cells(25, 2)
  End If
  If ValueV < Cells(7, 3).Value Then
    ValueV = Vincrement + Cells(26, 2)
  End If
  If ValueW < Cells(8, 3).Value Then
    ValueW = Wincrement + Cells(27, 2)
  End If


'Write current output vector.

If ValueX <= Cells(2, 3) Then
  Cells(32 + i, 2) = ValueX
Else
  If ValueX > Cells(2, 3) Then
    Cells(32 + i, 2) = 0
  End If
End If

If ValueY <= Cells(3, 3) Then
  Cells(32 + i, 3) = ValueY
Else
  If ValueY > Cells(3, 3) Then
    Cells(32 + i, 3) = 0
  End If
End If

If ValueZ <= Cells(4, 3) Then
  Cells(32 + i, 4) = ValueZ
Else
  If ValueZ > Cells(4, 3) Then
    Cells(32 + i, 4) = 0
  End If
End If

If ValueT <= Cells(5, 3) Then
  Cells(32 + i, 5) = ValueT
Else
  If ValueT > Cells(5, 3) Then
    Cells(32 + i, 5) = 0
  End If
End If

If ValueU <= Cells(6, 3) Then
  Cells(32 + i, 6) = ValueU
Else
  If ValueU > Cells(6, 3) Then
    Cells(32 + i, 6) = 0
  End If
End If

If ValueV <= Cells(7, 3) Then
  Cells(32 + i, 7) = ValueV
```

```
Else
   If ValueV > Cells(7, 3) Then
      Cells(32 + i, 7) = 0
   End If
End If

If ValueW <= Cells(8, 3) Then
   Cells(32 + i, 8) = ValueW
Else
   If ValueW > Cells(8, 3) Then
      Cells(32 + i, 8) = 0
   End If
End If

'Write to sort table.
Sortx = Cells(32, 2)
Sorty = Cells(32, 3)
Sortz = Cells(32, 4)
Sortt = Cells(32, 5)
Sortu = Cells(32, 6)
Sortv = Cells(32, 7)
Sortw = Cells(32, 8)

Cells(32, 10) = Sortx
Cells(33, 10) = Sorty
Cells(34, 10) = Sortz
Cells(35, 10) = Sortt
Cells(36, 10) = Sortu
Cells(37, 10) = Sortv
Cells(38, 10) = Sortw

i = i + 1

Loop While ((ValueX < X2) Or (ValueY < Y2) Or (ValueZ < Z2) Or (ValueT < T2) Or (ValueU < U2) Or
(ValueV < V2) Or (ValueW < W2))

Cells(2, 11) = i
'


End Sub
```

```
Sub SORT()
'
' SORT Macro
' Macro recorded 4/23/2001 by rantoine
'

'
   Range("J32:J38").Select
   Selection.SORT Key1:=Range("J32"), Order1:=xlDescending, Header:=xlGuess _
      , OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
End Sub
```

```
Sub Filler()
'
' Filler Macro
' Macro recorded 4/11/2001 by rantoine
'
Dim Xi As Integer
Dim Yi As Integer
Dim Zi As Integer
Dim Ti As Integer
Dim Ui As Integer
Dim Vi As Integer
Dim Wi As Integer

Dim Xcomp As Double
Dim Ycomp As Double
Dim Zcomp As Double
Dim Tcomp As Double
Dim Ucomp As Double
Dim Vcomp As Double
Dim Wcomp As Double

Dim FILL As Integer
Dim Fillin As Integer

Xi = Cells(2, 2).Value
Yi = Cells(3, 2).Value
Zi = Cells(4, 2).Value
Ti = Cells(5, 2).Value
Ui = Cells(6, 2).Value
Vi = Cells(7, 2).Value
Wi = Cells(8, 2).Value

Xcomp = Cells(21, 2).Value
Ycomp = Cells(22, 2).Value
Zcomp = Cells(23, 2).Value
Tcomp = Cells(24, 2).Value
Ucomp = Cells(25, 2).Value
Vcomp = Cells(26, 2).Value
Wcomp = Cells(27, 2).Value


FILL = Cells(2, 11).Value
a = 0
b = 0
c = 0

For Fillin = 1 To FILL

  Do

    Xtemp = Cells(32 + b, 13).Value
    Ytemp = Cells(32 + b, 14).Value
```

```
Ztemp = Cells(32 + b, 15).Value
Ttemp = Cells(32 + b, 16).Value
Utemp = Cells(32 + b, 17).Value
Vtemp = Cells(32 + b, 18).Value
Wtemp = Cells(32 + b, 19).Value

If ((Xtemp - Xi) > 0) And (Xcomp = Cells(32 + c, 10)) Then
  Xi = 1 + Xi
  ' Write x increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If ((Ytemp - Yi) > 0) And (Ycomp = Cells(32 + c, 10)) Then
  Yi = 1 + Yi
  ' Write y increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If ((Ztemp - Zi) > 0) And (Zcomp = Cells(32 + c, 10)) Then
  Zi = 1 + Zi
  ' Write z increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If ((Ttemp - Ti) > 0) And (Tcomp = Cells(32 + c, 10)) Then
  Ti = 1 + Ti
  ' Write t increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If ((Utemp - Ui) > 0) And (Ucomp = Cells(32 + c, 10)) Then
  Ui = 1 + Ui
  ' Write u increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If ((Vtemp - Vi) > 0) And (Vcomp = Cells(32 + c, 10)) Then
  Vi = 1 + Vi
  ' Write v increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If ((Wtemp - Wi) > 0) And (Wcomp = Cells(32 + c, 10)) Then
  Wi = 1 + Wi
  ' Write w increment fill coordinate.
  Cells(32 + a, 22) = Xi
  Cells(32 + a, 23) = Yi
  Cells(32 + a, 24) = Zi
  Cells(32 + a, 25) = Ti
  Cells(32 + a, 26) = Ui
  Cells(32 + a, 27) = Vi
  Cells(32 + a, 28) = Wi
  a = a + 1

End If

If c > (Cells(2, 4) - 1) Then
  c = 0
Else
  c = c + 1
```

```
   End If

  Loop While ((Xi < Xtemp) Or (Yi < Ytemp) Or (Zi < Ztemp) Or (Ti < Ttemp) Or (Ui < Utemp) Or (Vi <
Vtemp) Or (Wi < Wtemp))
c = 0
b = b + 1

Next Fillin

'Number of iterations after fill.
Cells(30, 27) = a

'Integer line length.
Cells(32, 53) = a
'
End Sub
```

```
Sub BuildSlopeAndError()
'
' BuildSlopeAndError Macro
' Macro recorded 4/23/2001 by rantoine
'

'


Dim Start As Double
Dim Finish As Double
Dim TotalTime As Double
Dim SlopeProgram As Integer

n = 0

Start = Timer

For SlopeProgram = 1 To 1
      Application.Run "RSArow.xls!RelSlope"
      Application.Run "RSArow.xls!SORT"
      Application.Run "RSArow.xls!Filler"
      n = n + 1
Next SlopeProgram

'Calculate Elapsed Time
   Finish = Timer    ' Set end time.
   TotalTime = Finish - Start    ' Calculate total time.

'Write program time to excel
Cells(29, 4) = TotalTime
Cells(32, 57) = TotalTime

MsgBox "Program time is " & TotalTime & " seconds"

   Application.Run "RSArow.xls!RSAError"
End Sub
```

Appendix E: CSA SOURCE CODE

```
Sub RelSlope()
'
' RelSlope Macro
' Macro recorded 4/9/2001 by USAF
'
Dim ValueX As Double
Dim ValueY As Double
Dim ValueZ As Double
Dim ValueT As Double
Dim ValueU As Double
Dim ValueV As Double
Dim ValueW As Double

Dim X2 As Integer
Dim Y2 As Integer
Dim Z2 As Integer
Dim T2 As Integer
Dim U2 As Integer
Dim V2 As Integer
Dim W2 As Integer

Dim i As Integer
Dim n As Integer
Dim z As Integer

Dim num As Integer
Dim Denominator As Integer
Dim Biggest As Integer
Dim Increment As Double

Dim Xincrement As Double
Dim Yincrement As Double
Dim Zincrement As Double
Dim Tincrement As Double
Dim Uincrement As Double
Dim Vincrement As Double
Dim Wincrement As Double

Dim Sortx As Double
Dim Sorty As Double
Dim Sortz As Double
Dim Sortt As Double
Dim Sortu As Double
Dim Sortv As Double
Dim Sortw As Double

'Get initial starting values
ValueX = Cells(2, 2).Value
ValueY = Cells(3, 2).Value
ValueZ = Cells(4, 2).Value
```

```
ValueT = Cells(5, 2).Value
ValueU = Cells(6, 2).Value
ValueV = Cells(7, 2).Value
ValueW = Cells(8, 2).Value

'Get final values
X2 = Cells(2, 3).Value
Y2 = Cells(3, 3).Value
Z2 = Cells(4, 3).Value
T2 = Cells(5, 3).Value
U2 = Cells(6, 3).Value
V2 = Cells(7, 3).Value
W2 = Cells(8, 3).Value

z = 0
num = 0

For Biggest = 1 To 7
Denominator = Cells(10 + z, 2).Value
If Denominator > num Then
   num = Denominator
   Cells(18, 2) = num
End If

z = z + 1

Next Biggest

i = 0
n = 0

Do

Xincrement = ValueX
Yincrement = ValueY
Zincrement = ValueZ
Tincrement = ValueT
Uincrement = ValueU
Vincrement = ValueV
Wincrement = ValueW

'Now increment values according to slope ratios
If ValueX < Cells(2, 3).Value Then
   ValueX = Xincrement + Cells(21, 2)
End If
If ValueY < Cells(3, 3).Value Then
   ValueY = Yincrement + Cells(22, 2)
End If
If ValueZ < Cells(4, 3).Value Then
   ValueZ = Zincrement + Cells(23, 2)
End If
If ValueT < Cells(5, 3).Value Then
   ValueT = Tincrement + Cells(24, 2)
End If
```

```
If ValueU < Cells(6, 3).Value Then
  ValueU = Uincrement + Cells(25, 2)
End If
If ValueV < Cells(7, 3).Value Then
  ValueV = Vincrement + Cells(26, 2)
End If
If ValueW < Cells(8, 3).Value Then
  ValueW = Wincrement + Cells(27, 2)
End If


'Write current output vector.

If ValueX <= Cells(2, 3) Then
  Cells(32 + i, 2) = ValueX
Else
  If ValueX > Cells(2, 3) Then
    Cells(32 + i, 2) = 0
  End If
End If

If ValueY <= Cells(3, 3) Then
  Cells(32 + i, 3) = ValueY
Else
  If ValueY > Cells(3, 3) Then
    Cells(32 + i, 3) = 0
  End If
End If

If ValueZ <= Cells(4, 3) Then
  Cells(32 + i, 4) = ValueZ
Else
  If ValueZ > Cells(4, 3) Then
    Cells(32 + i, 4) = 0
  End If
End If

If ValueT <= Cells(5, 3) Then
  Cells(32 + i, 5) = ValueT
Else
  If ValueT > Cells(5, 3) Then
    Cells(32 + i, 5) = 0
  End If
End If

If ValueU <= Cells(6, 3) Then
  Cells(32 + i, 6) = ValueU
Else
  If ValueU > Cells(6, 3) Then
    Cells(32 + i, 6) = 0
  End If
End If

If ValueV <= Cells(7, 3) Then
```

```
  Cells(32 + i, 7) = ValueV
Else
  If ValueV > Cells(7, 3) Then
    Cells(32 + i, 7) = 0
  End If
End If

If ValueW <= Cells(8, 3) Then
  Cells(32 + i, 8) = ValueW
Else
  If ValueW > Cells(8, 3) Then
    Cells(32 + i, 8) = 0
  End If
End If

'Write to sort table.
Sortx = Cells(32, 2)
Sorty = Cells(32, 3)
Sortz = Cells(32, 4)
Sortt = Cells(32, 5)
Sortu = Cells(32, 6)
Sortv = Cells(32, 7)
Sortw = Cells(32, 8)

Cells(32, 10) = Sortx
Cells(33, 10) = Sorty
Cells(34, 10) = Sortz
Cells(35, 10) = Sortt
Cells(36, 10) = Sortu
Cells(37, 10) = Sortv
Cells(38, 10) = Sortw

i = i + 1

Loop While ((ValueX < X2) Or (ValueY < Y2) Or (ValueZ < Z2) Or (ValueT < T2) Or (ValueU < U2) Or
(ValueV < V2) Or (ValueW < W2))

Cells(2, 11) = i
'


End Sub
```

Note: The Sort and Filler subroutines are the same as RSA source code.

```
Sub BuildSlopeAndError()
'
' BuildSlopeAndError Macro
' Macro recorded 4/23/2001 by rantoine
'

'
Dim Start As Double
Dim Finish As Double
Dim TotalTime As Double
Dim SegmentProgram As Integer

n = 0

Start = Timer

For SegmentProgram = 1 To 1
   Application.Run "CSArow.xls!RelSlope"
   Application.Run "CSArow.xls!SORT"
   Application.Run "CSArow.xls!Filler"

Next SegmentProgram

'Calculate Elapsed Time
   Finish = Timer    ' Set end time.
   TotalTime = Finish - Start    ' Calculate total time.

'Write program time to excel
Cells(29, 4) = TotalTime
Cells(32, 57) = TotalTime
MsgBox "Program time is  " & TotalTime & " seconds"

   Application.Run "CSArow.xls!RSAError"
End Sub
```

Appendix F: SCA SOURCE CODE

```
Sub SlopeCalculation()
'
' SlopeCalculation Macro
' Macro recorded 3/12/01 by Capt. Rich Antoine
'

'


Dim ValueX As Integer
Dim ValueY As Integer
Dim ValueZ As Integer
Dim ValueT As Integer
Dim ValueU As Integer
Dim ValueV As Integer
Dim ValueW As Integer
Dim X2 As Integer
Dim Y2 As Integer
Dim Z2 As Integer
Dim T2 As Integer
Dim U2 As Integer
Dim V2 As Integer
Dim W2 As Integer
Dim i As Integer
Dim n As Integer
Dim Dx As Double
Dim Dy As Double
Dim Dz As Double
Dim Dt As Double
Dim Du As Double
Dim Dv As Double
Dim Dw As Double
Dim BestDim1 As Double
Dim BestDim2 As Double
Dim SlopeCompare As Integer
Dim ClosestSlope As Double
Dim NewDim1 As Double
Dim NewDim2 As Double
Dim NewSlope As Double

Dim ValueIncrement1 As Integer
Dim ValueIncrement2 As Integer


i = 0
NewSlope = 0
BestDim1 = 0
BestDim2 = 0

'Get initial starting values
ValueX = Cells(2, 2).Value
```

```
ValueY = Cells(3, 2).Value
ValueZ = Cells(4, 2).Value
ValueT = Cells(5, 2).Value
ValueU = Cells(6, 2).Value
ValueV = Cells(7, 2).Value
ValueW = Cells(8, 2).Value

'Get final values
X2 = Cells(2, 3).Value
Y2 = Cells(3, 3).Value
Z2 = Cells(4, 3).Value
T2 = Cells(5, 3).Value
U2 = Cells(6, 3).Value
V2 = Cells(7, 3).Value
W2 = Cells(8, 3).Value

Do

Dx = X2 - ValueX
Dy = Y2 - ValueY
Dz = Z2 - ValueZ
Dt = T2 - ValueT
Du = U2 - ValueU
Dv = V2 - ValueV
Dw = W2 - ValueW


'write Delta Values to Sheet
 Cells(10, 2) = Dx
 Cells(11, 2) = Dy
 Cells(12, 2) = Dz
 Cells(13, 2) = Dt
 Cells(14, 2) = Du
 Cells(15, 2) = Dv
 Cells(16, 2) = Dw

'Slope Comparison
ClosestSlope = 0
n = 0

For SlopeCompare = 1 To 42

NewDim1 = Cells(18 + n, 4)
NewDim2 = Cells(18 + n, 5)

If NewDim2 = 0 Then
    NewDim2 = NewDim2
Else
   NewSlope = NewDim1 / NewDim2

   If (NewSlope >= ClosestSlope) And (NewSlope <= 1) Then
      If Cells(18 + n, 5) > 0 Then
      BestDim1 = NewDim1
      BestDim2 = NewDim2
```

```
        ClosestSlope = BestDim1 / BestDim2
        End If
    End If
End If

n = n + 1

Next SlopeCompare


'Now Check Which Value Should Update

If (BestDim1 = Dx) And (BestDim2 = Dy) Then
    ValueIncrement1 = ValueX
    ValueIncrement2 = ValueY
    '1
    'ValueX = Int(ValueIncrement1 + NewSlope + 0.5)
    ValueY = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dy) And (BestDim2 = Dx) Then
        ValueIncrement1 = ValueY
        ValueIncrement2 = ValueX
        '2
        'ValueY = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueX = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dz) And (BestDim2 = Dy) Then
        ValueIncrement1 = ValueZ
        ValueIncrement2 = ValueY
        '3
        'ValueZ = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueY = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dy) And (BestDim2 = Dz) Then
        ValueIncrement1 = ValueY
        ValueIncrement2 = ValueZ
        '4
        'ValueY = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueZ = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dt) And (BestDim2 = Dy) Then
        ValueIncrement1 = ValueT
        ValueIncrement2 = ValueY
        '5
        'ValueT = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueY = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dy) And (BestDim2 = Dt) Then
        ValueIncrement1 = ValueY
        ValueIncrement2 = ValueT
        '6
        'ValueY = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueT = (ValueIncrement2 + 1)
```

```
Else
   If (BestDim1 = Du) And (BestDim2 = Dy) Then
      ValueIncrement1 = ValueU
      ValueIncrement2 = ValueY
      '7
      'ValueU = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueY = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dy) And (BestDim2 = Du) Then
      ValueIncrement1 = ValueY
      ValueIncrement2 = ValueU
      '8
      'ValueY = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueU = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dv) And (BestDim2 = Dy) Then
      ValueIncrement1 = ValueV
      ValueIncrement2 = ValueY
      '9
      'ValueV = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueY = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dy) And (BestDim2 = Dv) Then
      ValueIncrement1 = ValueY
      ValueIncrement2 = ValueV
      '10
      'ValueY = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueV = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dw) And (BestDim2 = Dy) Then
      ValueIncrement1 = ValueW
      ValueIncrement2 = ValueY
      '11
      'ValueW = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueY = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dy) And (BestDim2 = Dw) Then
      ValueIncrement1 = ValueY
      ValueIncrement2 = ValueW
      '12
      'ValueY = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueW = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dt) And (BestDim2 = Dx) Then
      ValueIncrement1 = ValueT
      ValueIncrement2 = ValueX
      '13
      'ValueT = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueX = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dx) And (BestDim2 = Dt) Then
      ValueIncrement1 = ValueX
      ValueIncrement2 = ValueT
      '14
```

```
        'ValueX = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueT = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dz) And (BestDim2 = Dx) Then
        ValueIncrement1 = ValueZ
        ValueIncrement2 = ValueX
        '15
        'ValueZ = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueX = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dx) And (BestDim2 = Dz) Then
        ValueIncrement1 = ValueX
        ValueIncrement2 = ValueZ
        '16
        'ValueX = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueZ = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Du) And (BestDim2 = Dx) Then
        ValueIncrement1 = ValueU
        ValueIncrement2 = ValueX
        '17
        'ValueU = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueX = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dx) And (BestDim2 = Du) Then
        ValueIncrement1 = ValueX
        ValueIncrement2 = ValueU
        '18
        'ValueX = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueU = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dv) And (BestDim2 = Dx) Then
        ValueIncrement1 = ValueV
        ValueIncrement2 = ValueX
        '19
        'ValueV = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueX = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dx) And (BestDim2 = Dv) Then
        ValueIncrement1 = ValueX
        ValueIncrement2 = ValueV
        '20
        'ValueX = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueV = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dw) And (BestDim2 = Dx) Then
        ValueIncrement1 = ValueW
        ValueIncrement2 = ValueX
        '21
        'ValueW = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueX = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dx) And (BestDim2 = Dw) Then
        ValueIncrement1 = ValueX
        ValueIncrement2 = ValueW
        '22
        'ValueX = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueW = (ValueIncrement2 + 1)

Else
    If (BestDim1 = Dt) And (BestDim2 = Dz) Then
        ValueIncrement1 = ValueT
        ValueIncrement2 = ValueZ
        '23
        'ValueT = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueZ = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dz) And (BestDim2 = Dt) Then
        ValueIncrement1 = ValueZ
        ValueIncrement2 = ValueT
        '24
        'ValueZ = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueT = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Du) And (BestDim2 = Dz) Then
        ValueIncrement1 = ValueU
        ValueIncrement2 = ValueZ
        '25
        'ValueU = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueZ = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dz) And (BestDim2 = Du) Then
        ValueIncrement1 = ValueZ
        ValueIncrement2 = ValueU
        '26
        'ValueZ = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueU = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dv) And (BestDim2 = Dz) Then
        ValueIncrement1 = ValueV
        ValueIncrement2 = ValueZ
        '27
        'ValueV = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueZ = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dz) And (BestDim2 = Dv) Then
        ValueIncrement1 = ValueZ
        ValueIncrement2 = ValueV
        '28
        'ValueZ = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueV = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dw) And (BestDim2 = Dz) Then
        ValueIncrement1 = ValueW
        ValueIncrement2 = ValueZ
        '29
        'ValueW = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueZ = (ValueIncrement2 + 1)
```

```
Else
   If (BestDim1 = Dz) And (BestDim2 = Dw) Then
      ValueIncrement1 = ValueZ
      ValueIncrement2 = ValueW
      '30
      'ValueZ = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueW = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Du) And (BestDim2 = Dt) Then
      ValueIncrement1 = ValueU
      ValueIncrement2 = ValueT
      '31
      'ValueU = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueT = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dt) And (BestDim2 = Du) Then
      ValueIncrement1 = ValueT
      ValueIncrement2 = ValueU
      '32
      'ValueT = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueU = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dv) And (BestDim2 = Dt) Then
      ValueIncrement1 = ValueV
      ValueIncrement2 = ValueT
      '33
      'ValueV = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueT = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dt) And (BestDim2 = Dv) Then
      ValueIncrement1 = ValueT
      ValueIncrement2 = ValueV
      '34
      'ValueT = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueV = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dw) And (BestDim2 = Dt) Then
      ValueIncrement1 = ValueW
      ValueIncrement2 = ValueT
      '35
      'ValueW = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueT = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dt) And (BestDim2 = Dw) Then
      ValueIncrement1 = ValueT
      ValueIncrement2 = ValueW
      '36
      'ValueT = Int(ValueIncrement1 + NewSlope + 0.5)
      ValueW = (ValueIncrement2 + 1)
Else
   If (BestDim1 = Dv) And (BestDim2 = Du) Then
      ValueIncrement1 = ValueV
      ValueIncrement2 = ValueU
      '37
```

```
        'ValueV = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueU = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Du) And (BestDim2 = Dv) Then
        ValueIncrement1 = ValueU
        ValueIncrement2 = ValueV
        '38
        'ValueU = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueV = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dw) And (BestDim2 = Du) Then
        ValueIncrement1 = ValueW
        ValueIncrement2 = ValueU
        '39
        'ValueW = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueU = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Du) And (BestDim2 = Dw) Then
        ValueIncrement1 = ValueU
        ValueIncrement2 = ValueW
        '40
        'ValueU = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueW = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dw) And (BestDim2 = Dv) Then
        ValueIncrement1 = ValueW
        ValueIncrement2 = ValueV
        '41
        'ValueW = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueV = (ValueIncrement2 + 1)
Else
    If (BestDim1 = Dv) And (BestDim2 = Dw) Then
        ValueIncrement1 = ValueV
        ValueIncrement2 = ValueW
        '42
        'ValueV = Int(ValueIncrement1 + NewSlope + 0.5)
        ValueW = (ValueIncrement2 + 1)
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If

'write current output vector
Cells(65 + i, 2) = ValueX
Cells(65 + i, 3) = ValueY
Cells(65 + i, 4) = ValueZ
Cells(65 + i, 5) = ValueT
Cells(65 + i, 6) = ValueU
Cells(65 + i, 7) = ValueV
Cells(65 + i, 8) = ValueW

i = i + 1


Loop While ((ValueX < X2) Or (ValueY < Y2) Or (ValueZ < Z2) Or (ValueT < T2) Or (ValueU < U2) Or
(ValueV < V2) Or (ValueW < W2))

Cells(2, 11) = i



End Sub
Sub ErrorCalculation()
'
' ErrorCalculation Macro
' Macro recorded 3/12/01 by SC
'

'
```

```
Dim t As Double
Dim tv1 As Double
Dim tv2 As Double
Dim tv3 As Double
Dim tv4 As Double
Dim tv5 As Double
Dim tv6 As Double
Dim tv7 As Double
Dim n As Integer
Dim Tcalc As Integer
Dim NumSlope As Integer

Dim xdist As Double
Dim ydist As Double
Dim zdist As Double
Dim tdist As Double
Dim udist As Double
Dim vdist As Double
Dim wdist As Double

Dim Real_d As Double
Dim Integer_d As Double

Dim Po(1 To 7) As Double
Dim P1(1 To 7) As Double
Dim V(1 To 7) As Double
Dim R(1 To 7) As Double

Dim RminusPo1 As Double
Dim RminusPo2 As Double
Dim RminusPo3 As Double
Dim RminusPo4 As Double
Dim RminusPo5 As Double
Dim RminusPo6 As Double
Dim RminusPo7 As Double
Dim RminusPoxV As Double

Dim VxV As Double

Dim PoCurve1 As Double
Dim PoCurve2 As Double
Dim PoCurve3 As Double
Dim PoCurve4 As Double
Dim PoCurve5 As Double
Dim PoCurve6 As Double
Dim PoCurve7 As Double

Dim d As Double

'Origin Points
Po(1) = Cells(2, 7).Value
Po(2) = Cells(3, 7).Value
Po(3) = Cells(4, 7).Value
Po(4) = Cells(5, 7).Value
```

```
Dim t As Double
Dim tv1 As Double
Dim tv2 As Double
Dim tv3 As Double
Dim tv4 As Double
Dim tv5 As Double
Dim tv6 As Double
Dim tv7 As Double
Dim n As Integer
Dim Tcalc As Integer
Dim NumSlope As Integer

Dim xdist As Double
Dim ydist As Double
Dim zdist As Double
Dim tdist As Double
Dim udist As Double
Dim vdist As Double
Dim wdist As Double

Dim Real_d As Double
Dim Integer_d As Double

Dim Po(1 To 7) As Double
Dim P1(1 To 7) As Double
Dim V(1 To 7) As Double
Dim R(1 To 7) As Double

Dim RminusPo1 As Double
Dim RminusPo2 As Double
Dim RminusPo3 As Double
Dim RminusPo4 As Double
Dim RminusPo5 As Double
Dim RminusPo6 As Double
Dim RminusPo7 As Double
Dim RminusPoxV As Double

Dim VxV As Double

Dim PoCurve1 As Double
Dim PoCurve2 As Double
Dim PoCurve3 As Double
Dim PoCurve4 As Double
Dim PoCurve5 As Double
Dim PoCurve6 As Double
Dim PoCurve7 As Double

Dim d As Double

'Origin Points
Po(1) = Cells(2, 7).Value
Po(2) = Cells(3, 7).Value
Po(3) = Cells(4, 7).Value
Po(4) = Cells(5, 7).Value
Po(5) = Cells(6, 7).Value
Po(6) = Cells(7, 7).Value
Po(7) = Cells(8, 7).Value

'End Points

P1(1) = Cells(2, 8).Value
P1(2) = Cells(3, 8).Value
P1(3) = Cells(4, 8).Value
P1(4) = Cells(5, 8).Value
P1(5) = Cells(6, 8).Value
P1(6) = Cells(7, 8).Value
P1(7) = Cells(8, 8).Value

'Line Definition Vector, V = P1 - Po
V(1) = Cells(2, 9).Value
V(2) = Cells(3, 9).Value
V(3) = Cells(4, 9).Value
V(4) = Cells(5, 9).Value
V(5) = Cells(6, 9).Value
V(6) = Cells(7, 9).Value
V(7) = Cells(8, 9).Value

'Multiply V column vector times V column Vector

VxV = (V(1) * V(1)) + (V(2) * V(2)) + (V(3) * V(3)) + (V(4) * V(4)) + (V(5) * V(5)) + (V(6) * V(6)) +
(V(7) * V(7))

'Get the value of counter i from slope comparison at top spreadsheet and call it NumSlope

NumSlope = Cells(2, 11).Value


n = 0
For Tcalc = 1 To NumSlope
   R(1) = Cells(65 + n, 2)
   R(2) = Cells(65 + n, 3)
   R(3) = Cells(65 + n, 4)
   R(4) = Cells(65 + n, 5)
   R(5) = Cells(65 + n, 6)
   R(6) = Cells(65 + n, 7)
   R(7) = Cells(65 + n, 8)

'Now subtract Po column vector from R Column vector

   RminusPo1 = R(1) - Po(1)
   RminusPo2 = R(2) - Po(2)
   RminusPo3 = R(3) - Po(3)
   RminusPo4 = R(4) - Po(4)
   RminusPo5 = R(5) - Po(5)
   RminusPo6 = R(6) - Po(6)
   RminusPo7 = R(7) - Po(7)

'Now multiply RminusPo Column Vector times the V Column Vector
```

```
RminusPoxV = (RminusPo1 * V(1)) + (RminusPo2 * V(2)) + (RminusPo3 * V(3)) + (RminusPo4 * V(4))
+ (RminusPo5 * V(5)) + (RminusPo6 * V(6)) + (RminusPo7 * V(7))

'Now divide RminusPoxV by VxV to get a value for t
t = (RminusPoxV / VxV)

'Now multiply t times v

tv1 = t * V(1)
tv2 = t * V(2)
tv3 = t * V(3)
tv4 = t * V(4)
tv5 = t * V(5)
tv6 = t * V(6)
tv7 = t * V(7)

'Now create the on curve solution point to be used in the error (Po + tv = 0)

PoCurve1 = tv1 + Po(1)
PoCurve2 = tv2 + Po(2)
PoCurve3 = tv3 + Po(3)
PoCurve4 = tv4 + Po(4)
PoCurve5 = tv5 + Po(5)
PoCurve6 = tv6 + Po(6)
PoCurve7 = tv7 + Po(7)

'Write the on curve solution point to the spreadsheet

Cells(65 + n, 10) = PoCurve1
Cells(65 + n, 11) = PoCurve2
Cells(65 + n, 12) = PoCurve3
Cells(65 + n, 13) = PoCurve4
Cells(65 + n, 14) = PoCurve5
Cells(65 + n, 15) = PoCurve6
Cells(65 + n, 16) = PoCurve7

'Now calculate d (distance) from of curve point R to the on curve solution point.

d = ((((R(1) - PoCurve1) ^ 2) + ((R(2) - PoCurve2) ^ 2) + ((R(3) - PoCurve3) ^ 2) + ((R(4) - PoCurve4) ^
2) + ((R(5) - PoCurve5) ^ 2) + ((R(6) - PoCurve6) ^ 2) + ((R(7) - PoCurve7) ^ 2))) ^ (1 / 2)

'Write the value of d to the spreadsheet

Cells(65 + n, 18) = d
Cells(65 + n, 31) = d

'Calculate distance in each dimension (x,y,z,t,u,v,w).

xdist = ((R(1) - PoCurve1) ^ 2) ^ (1 / 2)
ydist = ((R(2) - PoCurve2) ^ 2) ^ (1 / 2)
zdist = ((R(3) - PoCurve3) ^ 2) ^ (1 / 2)
tdist = ((R(4) - PoCurve4) ^ 2) ^ (1 / 2)
udist = ((R(5) - PoCurve5) ^ 2) ^ (1 / 2)
```

```
vdist = ((R(6) - PoCurve6) ^ 2) ^ (1 / 2)
wdist = ((R(7) - PoCurve7) ^ 2) ^ (1 / 2)

'Write the individual dimension distances to the spreadsheet.

Cells(65 + n, 22) = xdist
Cells(65 + n, 23) = ydist
Cells(65 + n, 24) = zdist
Cells(65 + n, 25) = tdist
Cells(65 + n, 26) = udist
Cells(65 + n, 27) = vdist
Cells(65 + n, 28) = wdist

'Measure the real line distance from start point to end point.

Real_d = ((((Po(1) - PoCurve1) ^ 2) + ((Po(2) - PoCurve2) ^ 2) + ((Po(3) - PoCurve3) ^ 2) + ((Po(4) -
PoCurve4) ^ 2) + ((Po(5) - PoCurve5) ^ 2) + ((Po(6) - PoCurve6) ^ 2) + ((Po(7) - PoCurve7) ^ 2))) ^ (1 / 2)

Cells(65 + n, 32) = Real_d
Cells(65, 33) = NumSlope

  n = n + 1
Next Tcalc


End Sub
```

```
Sub BuildSlopeAndError()
'
' BuildSlopeAndError Macro
' Macro recorded 4/23/2001 by rantoine
'

'


Dim Start As Double
Dim Finish As Double
Dim TotalTime As Double
Dim SlopeProgram As Integer

n = 0

Start = Timer

For SlopeProgram = 1 To 1
   Application.Run "SCArow.xls!SlopeCalculation"
   n = n + 1

Next SlopeProgram

'Calculate Elapsed Time
   Finish = Timer    ' Set end time.
   TotalTime = Finish - Start    ' Calculate total time.
```

```
'Write program time to excel
Cells(62, 2) = TotalTime
Cells(65, 37) = TotalTime
MsgBox "Program time is " & TotalTime & "  seconds"

   Application.Run "SCArow.xls!ErrorCalculation"
End Sub
```

Appendix G: Error Calculation Subroutine

```
Sub ErrorCalculation()
'
' ErrorCalculation Macro
' Macro recorded 3/12/01 by SC
'

'
Dim t As Double
Dim tv1 As Double
Dim tv2 As Double
Dim tv3 As Double
Dim tv4 As Double
Dim tv5 As Double
Dim tv6 As Double
Dim tv7 As Double
Dim n As Integer
Dim Tcalc As Integer
Dim NumSlope As Integer

Dim xdist As Double
Dim ydist As Double
Dim zdist As Double
Dim tdist As Double
Dim udist As Double
Dim vdist As Double
Dim wdist As Double

Dim Real_d As Double
Dim Integer_d As Double

Dim Po(1 To 7) As Double
Dim P1(1 To 7) As Double
Dim V(1 To 7) As Double
Dim R(1 To 7) As Double

Dim RminusPo1 As Double
Dim RminusPo2 As Double
Dim RminusPo3 As Double
Dim RminusPo4 As Double
Dim RminusPo5 As Double
Dim RminusPo6 As Double
Dim RminusPo7 As Double
Dim RminusPoxV As Double

Dim VxV As Double

Dim PoCurve1 As Double
Dim PoCurve2 As Double
Dim PoCurve3 As Double
Dim PoCurve4 As Double
Dim PoCurve5 As Double
Dim PoCurve6 As Double
Dim PoCurve7 As Double

Dim d As Double

'Origin Points
Po(1) = Cells(2, 7).Value
Po(2) = Cells(3, 7).Value
Po(3) = Cells(4, 7).Value
Po(4) = Cells(5, 7).Value
Po(5) = Cells(6, 7).Value
Po(6) = Cells(7, 7).Value
Po(7) = Cells(8, 7).Value

'End Points

P1(1) = Cells(2, 8).Value
P1(2) = Cells(3, 8).Value
P1(3) = Cells(4, 8).Value
P1(4) = Cells(5, 8).Value
P1(5) = Cells(6, 8).Value
P1(6) = Cells(7, 8).Value
P1(7) = Cells(8, 8).Value

'Line Definition Vector, V = P1 - Po
V(1) = Cells(2, 9).Value
V(2) = Cells(3, 9).Value
V(3) = Cells(4, 9).Value
V(4) = Cells(5, 9).Value
V(5) = Cells(6, 9).Value
V(6) = Cells(7, 9).Value
V(7) = Cells(8, 9).Value

'Multiply V column vector times V column Vector

VxV = (V(1) * V(1)) + (V(2) * V(2)) + (V(3) * V(3)) + (V(4) * V(4)) + (V(5) * V(5)) + (V(6) * V(6)) +
(V(7) * V(7))

'Get the value of counter i from slope comparison at top spreadsheet and call it NumSlope

NumSlope = Cells(2, 11).Value


n = 0
For Tcalc = 1 To NumSlope
   R(1) = Cells(65 + n, 2)
   R(2) = Cells(65 + n, 3)
   R(3) = Cells(65 + n, 4)
   R(4) = Cells(65 + n, 5)
   R(5) = Cells(65 + n, 6)
   R(6) = Cells(65 + n, 7)
   R(7) = Cells(65 + n, 8)

'Now subtract Po column vector from R Column vector
```

```
    RminusPo1 = R(1) - Po(1)
    RminusPo2 = R(2) - Po(2)
    RminusPo3 = R(3) - Po(3)
    RminusPo4 = R(4) - Po(4)
    RminusPo5 = R(5) - Po(5)
    RminusPo6 = R(6) - Po(6)
    RminusPo7 = R(7) - Po(7)

'Now multiply RminusPo Column Vector times the V Column Vector

RminusPoxV = (RminusPo1 * V(1)) + (RminusPo2 * V(2)) + (RminusPo3 * V(3)) + (RminusPo4 * V(4))
+ (RminusPo5 * V(5)) + (RminusPo6 * V(6)) + (RminusPo7 * V(7))

'Now divide RminusPoxV by VxV to get a value for t
t = (RminusPoxV / VxV)

'Now multiply t times v

tv1 = t * V(1)
tv2 = t * V(2)
tv3 = t * V(3)
tv4 = t * V(4)
tv5 = t * V(5)
tv6 = t * V(6)
tv7 = t * V(7)

'Now create the on curve solution point to be used in the error (Po + tv = 0)

PoCurve1 = tv1 + Po(1)
PoCurve2 = tv2 + Po(2)
PoCurve3 = tv3 + Po(3)
PoCurve4 = tv4 + Po(4)
PoCurve5 = tv5 + Po(5)
PoCurve6 = tv6 + Po(6)
PoCurve7 = tv7 + Po(7)

'Write the on curve solution point to the spreadsheet

Cells(65 + n, 10) = PoCurve1
Cells(65 + n, 11) = PoCurve2
Cells(65 + n, 12) = PoCurve3
Cells(65 + n, 13) = PoCurve4
Cells(65 + n, 14) = PoCurve5
Cells(65 + n, 15) = PoCurve6
Cells(65 + n, 16) = PoCurve7

'Now calculate d (distance) from of curve point R to the on curve solution point.

d = ((((R(1) - PoCurve1) ^ 2) + ((R(2) - PoCurve2) ^ 2) + ((R(3) - PoCurve3) ^ 2) + ((R(4) - PoCurve4) ^
2) + ((R(5) - PoCurve5) ^ 2) + ((R(6) - PoCurve6) ^ 2) + ((R(7) - PoCurve7) ^ 2))) ^ (1 / 2)

'Write the value of d to the spreadsheet

Cells(65 + n, 18) = d
Cells(65 + n, 31) = d

'Calculate distance in each dimension (x,y,z,t,u,v,w).

xdist = ((R(1) - PoCurve1) ^ 2) ^ (1 / 2)
ydist = ((R(2) - PoCurve2) ^ 2) ^ (1 / 2)
zdist = ((R(3) - PoCurve3) ^ 2) ^ (1 / 2)
tdist = ((R(4) - PoCurve4) ^ 2) ^ (1 / 2)
udist = ((R(5) - PoCurve5) ^ 2) ^ (1 / 2)
vdist = ((R(6) - PoCurve6) ^ 2) ^ (1 / 2)
wdist = ((R(7) - PoCurve7) ^ 2) ^ (1 / 2)

'Write the individual dimension distances to the spreadsheet.

Cells(65 + n, 22) = xdist
Cells(65 + n, 23) = ydist
Cells(65 + n, 24) = zdist
Cells(65 + n, 25) = tdist
Cells(65 + n, 26) = udist
Cells(65 + n, 27) = vdist
Cells(65 + n, 28) = wdist

'Measure the real line distance from start point to end point.

Real_d = ((((Po(1) - PoCurve1) ^ 2) + ((Po(2) - PoCurve2) ^ 2) + ((Po(3) - PoCurve3) ^ 2) + ((Po(4) -
PoCurve4) ^ 2) + ((Po(5) - PoCurve5) ^ 2) + ((Po(6) - PoCurve6) ^ 2) + ((Po(7) - PoCurve7) ^ 2))) ^ (1 / 2)

Cells(65 + n, 32) = Real_d
Cells(65, 33) = NumSlope

  n = n + 1
Next Tcalc


End Sub
```

Note: Error Calculation subroutine was modified to fit each algorithm.

Bibliography

Arya, Jagdish C. and Lardner, Robin W. College Algebra with Applications. New Jersey: Prentice-Hall, 1983.

Albright, S. Christian. VBA for Modelers. Pacific Grove CA: Duxbury Thomas Learning, 2001.

Baker, William P. Assistant Professor of Logistics, Mathematics & Statistics, Air Force Institute of Technology, Wright-Patterson AFB OH. Personal interview. 22 March 2001.

Boonin, Elisabeth. Using Excel Visual Basic for Applications. Indianapolis: Que, 1995.

Bresenham, J. E. "Algorithm for Computer Control of a Digital Plotter," IBM Systems Journal,4-1:25-30 (1965).

Bu-Qing, Su and Ding-Yuan, Liu. Computational Geometry: Curve and Surface Modeling. New York: Academic Press Inc., 1989.

Buzo, Christopher D. A Decision Support Tool to Aid Campaign Planners in Selecting Combat Aircraft for Theater Crisis. MS Thesis, AFIT/GEE/ENS/00M-02. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Clark, Dean. "A 2-D DDA Algorithm for Fast Image Scaling," Dr. Dobb's Journal, 22-4:46-49(April 1997).

Devore, Jay L. Probability and Statistics For Engineering and the Sciences 5th Edition Pacific Grove CA: Duxbury, 2000.

Earnshaw, R. A. Fundamenatal Algorithms for Computer Graphics. New York: Springer-Verlag, 1985.

Earnshaw, R. A. "Line Tracking with Incremental Plotters," Computer Journal, 23:46-52 (November 1978).

Foley, James D. and Others. Computer Graphics: Principle and Practice 2nd Edition. NewYork: Addison-Wesley Publishing Company, 1990.

Hearn, Donald and Baker, M. Pauline. Computer Graphics. New Jersey: Prentice Hall, 1997.

Jaccobs, Timothy M. Assistant Professor of Computer Engineering, Dept. Electrical and Computer Engineering, Air Force Institute of Tech, Wright-Patterson AFB OH. Personal interview. 17 March 2001.

Johnson, A. W., Swartz, S. M., and Allen, C. M. "AFIT/ AEF: Mission-Resource Value Assessment Technique," DARPA/ISO Advanced Logistics Project Workshop VII. Washington DC, 2000.

Newman, William M. and Sproull, Robert F. Principles of Interactive Computer Graphics 2nd Edition. New York: McGraw-Hill Book Company, 1979.

Pavlidis, Theo. Algorithms for Graphics and Image Processing. Rockville, MD: Computer Sciences Press Inc., 1982.

Perry, Greg and Hettihewa, Sanjaya. SAMS Teach Yourself Visual Basic 6 in 24 Hours. Indianapolis, SAMS MacMillan Computer Publishing, 1998.

Shamos, Michael Ian and Preparata, Franco P. Computational Geometry: An Introduction. New York: Springer-Verlag, 1990.

Stockton, F. G. "Algorithm 162: X-Move Plotting," Communications of the ACM,6-4:161 (April 1963).

Swartz, Stephen. "ALP Pilot Problem and Derivation of Mathematical Model." Unpublished report. Wright-Patterson AFB OH, 1999.

Swartz, Stephen M. Assistant Professor of Logistics, Dept. Operational Sciences, Air Force Institute of Tech, Wright-Patterson AFB OH. Personal interview. 02 April 2001.

Wakefield, David J. Identification of Preferred Operational Plan Force Mixes Using a Multiobjective Methodology to Optimize Resource Suitability and Lift Cost. MS Thesis, AFIT/GLM/ENS/01M-24. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2001.

Thomas, Edward. MRVAT Software Developer. Personal interview. 02 April 2001.

Walkenbach, John. Excel 2000 Power Programming with VBA. Chicago: IDG Books World Wide, Inc., 1999.

Williams, H.P. Model Building in Mathematical Programming2nd Edition. New York: John Wiley & Sons, 1985.

Xiang, Zhigang and Plastock, Roy. Computer Graphic 2nd Edition. New York: McGraw Hill, 2000.

Vita

On 16 Dec 1994, Richard Antoine graduated with a B.S. in Mechanical Engineering from Tuskegee University. Richard was commissioned as a 2nd Lieutenant in the United States Air Force on the same day.

2nd Lieutenant Antoine reported to the 846th Test Squadron, 46th Test Group at Holloman AFB, New Mexico in March 1995. There he served as a Rocket Sled Test Engineer. He performed numerous countermeasure, GPS and technology development sled tests.

In the summer of 1996, 2nd Lieutenant Antoine was selected for the NASA/DoD exchange program. On 13 September of 1996, Lieutenant Antoine reported to Dryden Flight Research Center were he served as Flight Test Operations Engineer on the F-15 ACTIVE program. After the one year assignment ended, 1st Lieutenant Antoine reported to the 412th Test Wing at Edwards AFB, California. There he was matrixed to the F-22 Combined Test Force. Richard served as a Structures Flight Test Engineer were he supported numerous F-22 flight test sorties.

In August of 1999, Captain Antoine was accepted and assigned to the Air Force Institute of Technology. After graduating in May 2001, he will be assigned to the C-17 System Program Office at Wright-Patterson AFB. He will serve as an Integrated Logistic Support Manager matrixed to the Air Vehicle IPT.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 14-05-2001	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* 1 Mar 2000 – 14 May 2001
4. TITLE AND SUBTITLE INTEGER APPROXIMATION OF REAL VALUED PREFERENCE CURVES		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Antoine, Richard M., Capt, USAF		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765		**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GLM/ENS/01J-01
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Todd Carrico DARPA/ISO 3701 North Fairfax Drive Arlington, Virginia 22203-1714 (703) 526-6616		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A primary challenge of the AFIT Mission Resource Value Assessment Tool is to approximate a given preference curve with integer valued mission ready resources. This thesis evaluated four candidate methods of accomplishing this approximation.

The thesis evaluated the implementation of the integer estimation approximation from a purely mathematical perspective. The models were measured against six quality and error measurement standards: convergence on an endpoint, convergence on any interior integer points, characterization of the overall error between the sequence of integer coordinates and the real valued linear function and characterization of the error in each individual dimension of the problem space. Finally, computer processing time was measured and a comparison of the lengths of the real valued linear function and the sequence of integer coordinates used to approximate the function were compared.

Based on these measures the Relative Slope Algorithm (RSA) was selected. RSA demonstrated the minimal error and consistently quick processing time. This algorithm will improve the Mission Resource Value Assessment Tool and further its impact on the Advanced Logistic project.

15. SUBJECT TERMS
Line Drawing, Linear Approximation, Preference Curve, Point Fitting, Algorithms, Integer Approximation, Resource Allocation, Integer Points, Real Valued Linear Curve

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Lt Col Alan W. Johnson
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	117	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4284 Alan.Johnson@afit.af.edu